AFIT/GE/ENG/99M-22

Structural Emergence and the Collaborative Behavior of Autonomous Nano-Satellites

THESIS

Daniel J. Petrovich, B.S.E.E
Lieutenant, USAF

AFIT/GE/ENG/99M-22

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 1999	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE
STRUCTURAL EMERGENCE AND THE COLLABORATIVE BEHAVIOR OF NANO-SATELLITES

5. FUNDING NUMBERS

6. AUTHOR(S)
Daniel J. Petrovich, Lieutenant, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology,
2950 P Street
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GE/ENG/99M-22

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFRL/SNAT
Attn: Jim Morgan
Area B, Bldg 62,
2241 Avionics Circle
(937) 255-1491 x3328 /DSN 785-1491 x3328/ morganjs@sensors.wpafb.af.mil

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Dr. Steven Gustafson / 255-3636 x4598 /DSN 785-3636 x4598/ gustafs@afit.af.mil

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The collaborative behavior of nano-satellites in a zero-gravity environment is explored, and satellite characteristics are proposed that maximize constellation robustness and minimize manufacturing costs. Behavioral algorithms are proposed to facilitate both swarming and structural formation and are validated using the Structural Emergence Simulator (STEMS) Graphical User Interface (GUI). A payload of multiple satellites is placed in a zero-gravity environment and released to re-configure autonomously into a pre-designed structure. Data transmission between satellites is not permitted during the swarming phase of the structure formation mission. A binary behavior algorithm is invented that produces a direction and magnitude solution to the satellite control system. A second behavior algorithm, the four-post algorithm, is invented to facilitate structure formation behavior. This algorithm switches satellite transmitter channels, effectively altering the path of incident swarming satellites. The algorithm is subject to two constraints: rules must be evaluated and acted upon locally, and the structure architecture must be known. Structural emergence is realized: the binary and four-post algorithms facilitat endless transitions from a gaseous swarming state to a solid structural state. Two methods of conserving fuel are discovered Fuel saving of 38% are realized by setting thruster levels based upon environmental noise, and fuel savings of 45% are realized by seeding structural formation prior to swarm equilibrium. Finally, analysis indicates a proportional relationship between architecture complexity and structure formation half-life.

14. SUBJECT TERMS
Distributed Satellite System (DSS), Collaborative Behavior, Structural Emergence, Cellular Automata, Structure Formation, Robotic, Nano-Satellite, Autonomous Reconfiguration

15. NUMBER OF PAGES
234

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

Disclaimer

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the United States Government.

AFIT/GE/ENG/99M-22

Structural Emergence and the Collaborative Behavior of Autonomous Nano-Satellites

THESIS

Presented to the Faculty of the Graduate School of Engineering of the Air Force Institute of Technology Air University In Partial Fulfillment for the Degree of

Master of Science

Specialization in: Electrical Engineering

Daniel J. Petrovich, B.S.E.E
Lieutenant, USAF

Air Force Institute of Technology

Wright-Patterson AFB, Ohio

8 March 1999

Sponsored by the the Air Force Research Laboratory, Target Recognition Division.

AFIT/GE/ENG/99M-22

Structural Emergence and the Collaborative Behavior of Autonomous Nano-Satellites

Daniel J. Petrovich, B.S.E.E

Lieutenant, USAF

Approved:

STEVEN GUSTAFSON, P.h.D., Research Advisor
AFIT Department of Electrical Engineering

9 Mar 99

ANDREW TERZUOLI, P.h.D., Committee Member
AFIT Department of Electrical Engineering

9 Mar 99

Preface

This research explores the technology and behavioral rules required to enable a constellation of identical satellites to autonomously configure into a solid structure while in High Earth Orbit (HEO). The primary result is that such a mission can only be accomplished if the satellites have at least the following characteristics: (1) mobility (2) two analog communications channels (3) one duplex radio channel, and (4) a close proximity attraction mechanism. A MATLAB Graphical User Interface, the *Structural Emergence Simulator (STEMS)*, is developed, and behavior algorithms are designed that successfully model a complete autonomous structural re-configuration in zero-gravity.

Acknowledgments

My personal appreciation extends to several people who responded at my behest to abstract yet pertinent questions throughout the long trek that was this thesis. Dr. Steve Gustafson was there for me at every turn to lend a helping hand. His unique ability to uncover the bottom line and to focus on data visualization are the combination of ideals that make this document a good example of unique research. Mr. Jim Morgan, just down the road at building 620: I thank you for supporting and showing interest in what appeared at first to be an off-the-wall idea. Your suggestions are greatly appreciated and can be found to developed throughout this document. Dr. Terzuoli: you sat through EENG700 for the Nth year in a row and still managed to maintain your vitality. How you do it I will never know. There were times during my stay here at AFIT that I dropped by your office just to tap your boundless reserves of energy for myself. Perhaps, someday, I will have a chance to give it back.

To Major Temple, Carrolyn Mallory, Major Chilton, and to those officers and NCOs who directly and indirectly influenced the standard flow of space and time to my betterment, Thank You. Let me mention Dr. Brown, the one truly eclectic man I've met during these two years at AFIT. A man who was once quoted as saying "what is that game you play with your kids that has (N-1) chairs?" in reference to musical chairs.

Finally, I must thank those points-of-conglomeration, the Comedy-Centrals of the world. The Comm. Lab was not just a place, but a way of life. The inhabitants thereof brightened my days and stole my chair on more than one occasion. Almost all of you are welcome to camp in my backyard at any time. Finally, what thesis would be complete without mentioning the Waffle House and W.O. Wrights. Without you this document would be twice as long and half as good.

Table Of Contents

Page

List of Figures

Page

Page

Page

List of Tables

Page

AFIT/GE/ENG/99M-22

Abstract

This work describes novel research on autonomous nano-satellite structure formation. The collaborative behavior of nano-satellites in a zero gravity environment is explored, and satellite characteristics are proposed that maximize constellation robustness and minimize manufacturing costs. Identical satellites with impulse thrusters and two light-intensity transceivers demonstrate abilities to form complex lattice structures. A comprehensive MATLAB® simulation engine, the *Structural Emergence Simulator* (STEMS), is developed for experimentation. Behavior algorithms are proposed to facilitate both swarming and structural formation and are validated using STEMS.

A payload of multiple satellites is placed in a zero gravity environment and released to reconfigure into a pre-designed structure. Data transmission between (or to) satellites is not permitted during the swarming phase of the structural reconfiguration mission. A swarming behavior function, *the binary behavior algorithm,* is invented that presents a satellite direction and magnitude solution to the satellite control system. The interplay of social forces due to the binary algorithm results in satellite swarming and a quiescent state of spatial equilibrium. Attractive and repulsive tendencies generate group cohesion while maintaining freedom of movement.

A second behavior function, *the four-post algorithm,* is invented to facilitate structure formation behavior. This algorithm switches satellite transmission channels, effectively altering the path of incident swarming satellites. The algorithm is subject to two constraints: rules must be evaluated and acted upon locally, and the final structural form must be known. The binary and four-post algorithms facilitate endless transitions from a gaseous swarming phase to a solid lattice structural phase.

Two methods of conserving fuel are discovered. Fuel savings of 38% are realized by setting a minimum thruster threshold based upon environmental noise levels, and fuel savings of 45% are realized by seeding structural formation prior to swarm equilibrium. Finally, analysis indicates a correlation between architecture complexity and structure formation half-life.

Structural Emergence and the Collaborative Behavior of Autonomous Nano-Satellites

1. Introduction

1.1 Background

1.1.1 Cellular Automata

Cellular automata (CA) [15,19] are discrete dynamical systems in which local states are completely specified in terms of local information. Cellular automata are best imagined as cellular systems that alter the state of local cells, asynchronously or synchronously, based upon a function of the measured state of local neighbors. The most favored design is a two dimensional lattice, but it is not difficult to imagine an N-dimensional automata with cells of varied geometry. The laws implemented in an automaton are local and uniform by definition, and automata are inherently parallel devices, i.e., each state in an R_1 x R_2 lattice can be updated simultaneously.

Cellular automata were introduced in the 1940s by John von Neumann [50,51] after a suggestion by Stanislaw Ulam. The idea was to describe a device made of identical components and capable of realizing a specialized machine. Continued work by Konrad Zuse, Arthur Burks [15], John Holland, John Conway, Tommaso Toffolio, and Stephen Wolfram [56,57], to name a few, ultimately produced a number of practical implementations of well-developed theory. Wolfram successfully classified the emergent properties of chaotic systems in a series of papers on the *Theory and Applications of Cellular Automata* [56,57,58,31]. According to a concise description of automata by Wolfram [56,57] CA have five fundamental defining characteristics (see Table 1.1) and can be decomposed into four classes (Table 1.2) based on

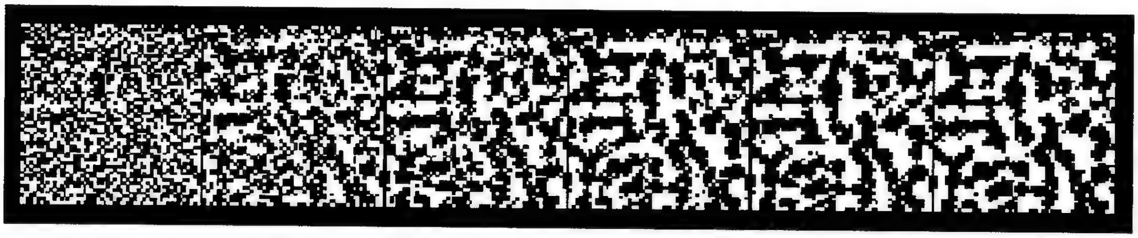


Figure 1.1 Deterministic Cellular Automata (DCA) at 1, 30K, 60K, 90K, 120K, 150K element modifications: *Gaseous Cohesion*

a spacio-temporal metrics. Langton argues that Wolfram's rule IV belongs naturally between rules II and III if CA are classified using established metrics of chaotic behavior. Wuensche [58] suggests that Wolfram's class I and II be combined naturally into a more concise *ordered (class 1-2), complex (class 4), and chaotic (class 3)* scheme. For the purpose of generating the proper abstract analogy for nano-satellites, we must only be aware that such classifications exist and that we seek an *ordered* quiescent state (Table 2, Rule 1).

Figure (1) is an automata coded in MATLAB® that models gaseous cohesion. It illustrates the transition of a chaotic system to one of lower spatial entropy. Iteration (1), Frame (1) [left] illustrates a random field of binary zeros and ones that represent water molecules in a diffuse state. A local rule that models molecular cohesion is applied iteratively and the result is a quiescent state analagous precipitation. However, the dimension of the resultant precipitate is a function of the local knowledge extent. For example, molecules in Figure (1) are affected by neighbors in a three pixel radius. However, if this radius of influence is extended and the same local rules applied, then resultant precipitates are of greater average dimension. The possible complexity of an end state is directly proportional to this knowledge extent. The nano-satellite structure formation implementation presented in this work has an extremely narrow local knowledge radius and by reducing hardware complexity, it limits the style of architecture. If this knowledge extent is too narrow, then it may be impossible to reach a desired end state.

List of Characteristics	
1	They consist of a discrete lattice of sites
2	They evolve in discrete time steps
3	Each site take on a finite set of possible values
4	The value of each site evolves according to the same deterministic rules
5	The rules for the evolution of a site depend only on a *local* neighborhood of sites around it

Table 1.1 Five fundamental characteristics of automata: Wolfram [57,58]

We seek a deterministic end state as a function of stochastic local rules, but only the initial conditions and not the local rules are known. One constraint imposed on this problem is that local rules must be functions of *local* neighbor states. Robustness and simplicity are compromised if global knowledge is shared. A solution to this problem is local behavior rule evolution using genetic algorithms, in which an initial rule is hypothesized and tested with mutations over successive generations until a solution is discovered. The problem with this approach is that no solution is guaranteed, and we must search a local rule space that suffers from massive dimensionality. Thus an algorithm must be developed to facilitate swarming and structure formation; to bridge the gap between known initial and final conditions while preserving locality.

1.1.2 Artificial Life

As carbon based life forms, we are naturally drawn to the study of carbon based life and hence to the field of biology. The biology of carbon based life [32] defines living systems as those that possess the following characteristics: (1) have highly organized bodily systems, (2) are chemically different from their environment, (3) take in energy from their environment, (4) respond to surrounding stimuli, (5) are particularly suited to their environment, and (6) can adapt to their surrounding environment [36,54]. These tenants are found to varying degrees in the progeny of modern man: silicon based machines. Although no machine exists (yet) that meets the most strict definition of life, there are robots that exhibit very life-like behavior.

Class	*Description*
I	A spatially homogenous state
II	A sequence of simple stable or periodic structures
III	Chaotic aperiodic behavior
IV	Complicated localized structures

Table 1.2 Cellular Automata classification: Wolfram [cite]

Animation, the ability to learn, intelligence, and the conversion of energy are all aspects of biological life that current robotic technologies can exhibit simultaneously. Self-reproduction, or the concept of Von Neumann universality, [14,16,50,51] on a physical level still eludes us. Until the silicon-based equivalent of meiosis and mitosis is achieved, the field of Artificial Life (AL or ALife) [7,38,41,42,43] must remain somewhat distant from the field of carbon based biology. Regardless of the classification, ALife remains a discipline that studies the properties of natural life by attempting to recreate biological phenomena using artificial media. Here 'artificial' is in the sense that the media is of a composition other than carbon-based molecules – a very human-centric notion. As described by Chris G. Langton [29,30]:

> **"ALife complements the traditional analytic approach of traditional biology with a synthetic approach in which, rather than studying biological phenomena by taking apart living organisms to see how they work, one attempts to put together systems that behave like living organisms."**

Although ALife is biologically inspired, the action of designing creatures with animalian or human characteristics is not new. The earliest mechanical devices that were capable of generating their own behavior were the early Egyptian water-clocks [7] called *Clepsydra.* They used the rate limiting process of dripping water to indicate the position of the sun. Mankind has a long history of attempting to map the mechanisms of this contemporary technology on to the workings of nature, trying to understand the latter in terms of the former. It is as if mankind has a predilection for re-instantiation in an attempt to surmount perceived inadequacies. If the next evolutionary step of an advanced species is replacement by hardware which that the *same* species designed, then humankind is well on the way to evolving out of its carbon-based shell. ALife is the study of this current (proposed) metamorphosis, just as biology is the study of the human mechanism. In keeping with the noble philosophy of ALife "to put together systems that behave like living organisms," the following pages explore one theoretical application of collaborative systems.

1.2 Original Mission

The original mission that motivated this thesis was, "to describe the simple local behavioral rules [45,46] that enable a robust constellation of satellites to swarm, then reconfigure into a pre-designed solid structure (May, 1997)." Three key words are: *simple*, *local*, and *pre-designed*, and each word carries important implications as indicated in the subsections. A new paradigm, with implementation of the recently promoted 'faster, better, cheaper' mantra at NASA, changed the primary metric by which spacecracft are judged from purely performance to 'specific performance' or performance per unit cost. The paradigm shift encouraged both a decrease in the cost and a decrease in the size of orbiting platforms. When launching an object into geosynchronous orbit costs $17,000 per pound, decreased size equates to decreased dollars spent. In partial response to this cost, MicroElectroMechanical Systems (MEMS) technology is thriving. MEMS devices are no longer laboratory curiosities: a large number of universities, companies, and nations have established laboratories and/or programs for research into the scientific fundamentals of such devices and their potential applications. For example, Germany recently completed a four year, $258 million project, and Japan is midway though a ten year, $171 million effort. With these new technologies, the bottom line remains reliability. To increase reliability locality must be increased; in other words, the division of labor must be evenly distributed. Here, labor is synonymous with sensing, computation, and motive action. An increase in locality is often a prerequisite for an increase in robustness. The merger of simplicity and locality decreases both orbital structure cost and the probability of failure.

Honeybees, for example, generate actions based upon local information. If the nest appears to be mis-shapen, then the nearest bee takes responsibility for fixing the *local* problem based upon decisions made in response to a mixture of pheromone [52], thermal, and visual cues [48]. No sooner has the bee modified a portion of the hive than it forgets the action entirely and moves on to the next menial task. How then does a swarm of honeybees create such a seemingly complex hive? The answer is that bees peform many iterations of simple local rules [9,11,12] and the result is an object that appears pre-designed. It should be noted that honeybees do not have an entire blueprint stored in memory, they are hard-wired to respond to stimuli [20, 24,52] based upon (what amounts to) stochastic local behavioral rules. Here 'hard-wired' does not imply non-adaptive or memoryless; however, reverse engineering (i.e., solving the inverse problem) a beehive or termite pillar and extracting the local rules required to make it is a difficult task and more than one set of local rules is likely to exist.

The inverse problem is described in terms of initial conditions, local rules, and Wolfram's CA classifications (Chapter 2, *Cellular Automata).* The inverse problem requires that we search a rule space for a set (of rules) that guarantees a known result, given a specified range of environmental conditions with ambient noise below some threshold. Humans are extraordinarily adept at determining what local actions must be taken to ensure a result, which introduces the concept of architecture [23]; the formal practice of generating global blueprints that workers (automata) are capable of executing. Thus, an architect mentally takes into account the local tasks workers must accomplish to construct a structure. Just as an architect is limited by the capabilities of the worker (and vice-versa), so too we are ultimately limited to a style of architecture; to single valued functions that describe surfaces in three-dimensions. From a practical standpoint, launching a constellation of satellites to construct an object in orbit requires the certainty of blueprints and it is desireable to decompose these blueprints into local tasks with a quick algorithm. Genetic algorithms may find a solution to the inverse problem, however, rule evolution can be a time consuming process. A means of decomposing global knowledge into completely local action is proposed in Chapter 4, *Methodology.* The trade-off for determining local rules given a set of blueprints is *increased speed* for a *decreased selection* of architectural styles.

1.4 Locality

Locality refers to the level at which computations are executed in a distributed network of processors. In this work it implies the act of information processing and reception on an individual satellite level. The ultimate goal is to implement distributed intelligence so that emergent behavior is realized:

> **"Emergence as a classical philosophical doctrine was (is) the belief that there will arise in complex systems new categories of behavior that cannot be derived from the system elements." [Boden 7]**

Thus, distributed systems that demonstrate emergent behavior are often mistaken for systems with highly-intelligent elements. Systems that demonstrate emergent behavior are capable of turning from chaotic behavior to yield functionality beyond that of any single element.

For example, behavior is based solely on local rules in a beehive. A veritable cornucopia of odors and imagery are received by a given bee, and actions is taken in the form of appendage and wingbeat movements. How fascinating that no digital Local Area Network (LAN), GPS, or wireless ethernet is ever used, yet bees accomplish their mission of hive construction with a high degree of success. Although no two beehives are identical, they are functional. Consider the antithesis of such Self-Organizing (SO) [13], behavior, the personal computer. The loss of even one transistor in a processor of millions can be catastrophic to the entire system. This problem may be addressed by distributing tasks to identical processors [19, 43].

Nearly every organism (on this planet) demonstrates either leaderless action or the ability to promote leaders without sacrificing the viability of the species. The sieve of evolution tends to favor locality as a means of avoiding the energy cost of higher intelligence. Creatures with more intelligence tend to demonstrate caste behavior and internal predation, because they develop the mental faculties consistent with an ability to design. If we extend this vein of abstraction to electronics, lower intelligence generally implies lower processor power, simplicity, and reduced cost. The expense is functionality, but only the functionality of *a single* element. This thesis is concerned with the net output of multiple intelligent agents, viz. satellites.

1.5 Scope

The MATLAB® Graphical User Interface (GUI) developed for this research allows a user to initialize and simulate the collaborative behavior of a nano-satellite constellation. As every satellite (agent) is physically and 'mentally' identical, initialization is performed in two steps. First, variables pertinent to the agent and the associated behavior function are defined. Next, initial ephemerides of the constellation are defined using a Generate Initial Conditions File (GICF) interface. This initial configuration is termed *a payload.*

Every model is imperfect; in our case the model is a discrete approximation of an inherently analog environment (reality). Certain assumptions (*Appendix A*) are made to increase the efficacy of the model and, in some instances, to simultaneously simplify the model code. However, at no point in the Structural Emergence Simulator (STEMS) code are *fudge factors,* i.e., linear or nonlinear adjustments (of a devious variety) designed to make otherwise paltry data appear commendable, added to improve the performance of the system. The laws of physics are always obeyed. In a number of instances, Additive White Gaussian Noise (AWGN) is injected to lessen the model/reality gap. Chapter 5 explores the effect of AWGN injection into the satellite sensor-to-thruster model. Results suggest that certain noise levels *improved* the overall performance of this system due to the annealing effect. Here, *annealing* is the process of disturbing solutions in regions of local minima to increase the probability of finding a global minimum. A model is only as good as its weakest assumption, so an effort is made to validate assumptions at every opportunity.

The fields of artificial life, cellular automata, and robotics are invoked at appropriate times to provide support via analogy. Thus, genetic algorithms and neural networks are elements of this thesis; and both are considered viable optimization tools. The correlation between this project and existing biological systems is no accident, because many millions of years of real-world evolution has repeatedly favored distributed cellular systems.

The effects of orbital dynamics are *not* taken into account in this model. Instead, an environment is defined in which motive agents (not necessarily satellites) are granted the ability to move in three dimensions while experiencing a uniform gravity gradient. Furthermore, agents experience no external forces (except those imposed by the exhuast plumes of other agents). This environment can be under-water, at a Lagrange point, or in an orbit far from a source of gravitation [1]. The choice of environment is only limited by the motive ability of a given agent. Space is selected as the natural environment to model collaborative behavior because it lacks atmosphere and objects in it obey simple Newtonian physics.

1.6 Outline

This thesis is organized into six chapters and two appendices. Chapter (1) describes the background and philosophy. Chapter (2) provides information on the experimental environment and introduces the conceptual satellite (agent) as modeled in STEMS. Chapter (3) describes two behavioral algorithms used to facilitate collaborative behavior and emphasizes the role of these algorithms in the STEM simulator. Chapter (4) formally describes the user interface and the file handling system and also describes how to set up STEMS up on your own computer. Chapter (5) presents results obtained with the proposed satellite model and associated behavior algorithms using the STEM simulator. Finally, Chapter (6) gives a brief description of results and provides recommendations for future research.

2. Environment

2.1 Coordinate System

The type of experimental environment determines the choice of coordinate system. The ideal nature of a zero gravity environment is required to validate many assumptions [*Appendix A*] made in this work, so a limited number of reference frames [40] may be applied. If we consider an environment distant from our solar system, the *Right ascension-declination coordinate (RADC) system* (Figure 2.1, left) is employed. In such a coordinate system, the position of an object in space is relative only to the universe or an infinite celestial sphere. It is centered on the Sun and aligned with the X axis along the Vernal Equinox, the Y axis in the plane of the celestial equator, and the Z axis pointing North. We must also be aware of the earth orbital reference frame, or the *Earth Centered Earth Fixed* (ECEF) coordinate system,

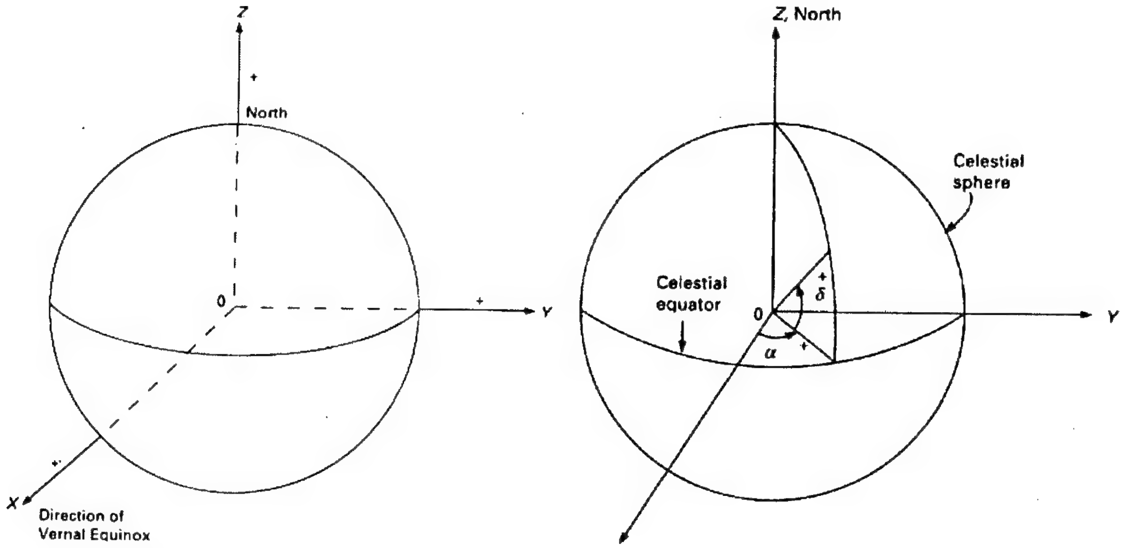


Figure 2.1 [Left] *Geocentric-Equatorial or Earth Centered Earth Fixed (ECEF) Coordinate System.* [Right] *Right ascension-declination coordinate system (RADC)*

(Figure 2.1, right) in which the equator is the fundamental plane and the geocenter constitutes the origin.

Since the Low Earth Orbit (LEO) and High Earth Orbit (HEO) orbital paths are not required in the STEMS model, two experimental reference frames are used: the *Structure CG Reference Frame* (SRF) and the *Collective Reference Frame* (CRF). The SRF centers the camera view on the structure center of gravity (CG) and rotates the structure about a vector $\overrightarrow{v_s}$ relative to that point. The CRF centers the model view on the continuously changing collective CG during the swarming phase of a mission. Both are used to view the motion of swarming agents within an arbitrary environment [*Appendix A*]. The SRF is centered on the forming structure center of mass (cg_s) and the CRF is centered on the joint structure/swarm center of mass (cg_o). Lastly, we must consider the Agent Reference (AR) Frame (see Chapter 4, *Methodology*). This frame is unique to each satellite and rotates in the CR and SR frames at a rate determined by the angular velocity of each satellite.

2.2 Forces

2.2.1 Orbital Altitude and Atmospheric Drag

In the near future (1999-2005), most nano-satellite constellations will find themselves in LEO [2,3,17]. Two major factors are involved in determining the likely altitude a fleet is likely to end up at: funding and timing. If a fleet is ready to fly and the opportunity to hitch-hike on a launch platform arises, then the payload has a better chance of flying. Constellations launched for the AFRL/TechSat 21 initiative [3,17] plan to use the Hitchhiker Pallet aboard NASA's space shuttle. The shuttle can place a payload into orbit at an altitude of 290 in 380 *km* with a satellite velocity of approximately $8,000\ \frac{m}{s}$. From Appendix A, *Assumptions,* it is readily apparent that we need an orbit far from earth to alleviate some of the complications of LEO. In LEO, atmospheric drag reduces the orbital lifetime of a satellite, which is generally detrimental. One exception is Dr. Frank Redd's Utah State University satellite, which intends to use the thin upper atmosphere to 'fly' the satellite into new orbits. USUsat's (15" x 5") 10kg satellite plans to use a 68332 micro controller with a tiny Motorola GPS receiver and permanent magnets for orientation control. Varying incident angles to the thin upper atmosphere provides an effective thrust vector that can be used for orbital transfer. For structure formation it is beneficial from an energy standpoint to be in a higher orbit. It is important to note that the velocity of a satellite in an elliptic orbit is

$$V^2 = \mu\left(\frac{2}{r} - \frac{1}{a}\right) \tag{1}$$

where μ is the universal gravitational parameter and equals $398,613.52\ km^3 \cdot s^{-2}$, r is the distance from center-of-mass earth to the center-of-mass satellite, and a is the length of the semi-major orbital axis. Atmospheric drag is proportional to V^2 and hence inversely proportional to the orbital radius. Therefore, orbital lifetime is greater in higher orbits due to exceedingly low atmospheric drag, and the satellite velocity tangent to the earth decreases with higher orbits. For lower orbital velocities, inter-satellite relative velocities are lower and fewer orbital corrections are required to maintain cohesion. Thus, we deduce that, 'collaborating satellites in higher orbits need be less motive than collaborating satellites in lower orbits.' From a cost standpoint the question becomes, 'is it cheaper to launch the constellation into a higher orbit and give it less net power, or is it cheaper to launch the fleet in to a lower orbit and give more net motive ability with an associated decrease in lifetime due to atmospheric drag?' Based upon presentations at the 1999 Air Force Nano-Satellite Collaborative Behavior Conference [3] at Kirtland AFB, New Mexico, the most probable option for nano-satellite orbits is LEO. However, rides-of-opportunity (i.e., leftover spaces on board a launch vehicle in which small payloads can be tucked away) are ever-increasing. When satellite constellations become *essential* (rather than *experimental),* then the funding will exist to launch only one payload – a massive constellation.

2.3 Communication for Collaboration, Command, and Control

Line-of-sight (LS) communication in a collective environment can serve two purposes. First, a break in data transfer suggests (to the receiver) that the transmitter was shadowed by a satellite or the structure as a whole. Because of shadowing, a satellite is immediately aware that a new object is closer and demands attention. Thus, transmitter data priority is partially determined by accidental shadowing. Second, solid LS communication grants the satellite *apriori* information that a current direct flight path exists between the transmitter and the receiver satellites. This path may not exist for long, but it can be used in conjunction with distance and/or Doppler information to optimize the receiver trajectory.

Any number of low frequency bands are suitable for broadcast radio communication intended for the group as a whole. Broadcast communication is used for synchronization, leadership assignment, and parameter upgrades. It is assumed that every satellite in the cluster and structure receives nearly 100% of all broadcast messages. However, there is a finite probability that one or more satellites will suffer from catastrophic bit error and not receive a broadcast message. Electromagnetic communication error rates increase with noise or positive attenuation and decrease with additional signal power, error correction hardware, and/or bandwidth. If a satellite should fail to receive a critical broadcast message, then it is considered a mutation and may compromise the structure formation mission. In STEMS no post-swarming radio broadcast messages are *ever* employed or required, so mutation is considered highly improbable.

2.4 Space Environment

2.4.1 Solar Radiation and the Solar Cycle

The majority of solar radiation in the near-earth region is from our own sun. Most of the sun's energy arrives at a relatively constant rate [39] in the form of low-energy photons. The remainder of solar radiation arrives at higher frequencies in the electromagnetic spectrum. The magnitude of this high frequency radiation varies with time according to an 11 year solar cycle. At times, intense bursts of charged particles are released from the surface of the sun during periods of solar particle events or 'solar flares.' The Zurich index is the most common solar activity metric. It measures the number of sunspots and sunspot groups observed on the surface of the sun and also takes into account a correction factor for observation error. A method exists for determining the Zurich sunspot number, but this method was improved by NASA's (TM-82478) equation for determining the smoothed solar flux data $\overline{F}_{10.7}$ given the smoothed sunspot term R_Z.

$$F_{10.7} \quad = \quad 49.40 + 0.97R_Z + 17.6e^{(-0.035R_Z)} \tag{2}$$

We are concerned with solar flux magnitude (Equation 2. 2), because it directly effects the reliability of earth-satellite communication. If the satellite constellation uses GPS, cell-phone technology, or wireless ethernet, $(\frac{E_b}{N_o})$ may exceed the error correcting ability of our system and thus jeopardize the formation flying mission. If the formation flying mission is influenced by a source of solar radiation, then appropriate steps must be taken to eliminate the effects to ensure mission success.

2.4.2 Atmospheric Drag

Larson and Wertz [39] formulated a handy mean/maximum orbital decay rate equation that returns the estimated decay in orbital altitude in kilometers per year,

$$orbit\_decay\_rate = \frac{-2\pi B \rho r^2}{P}, \tag{3}$$

using the Ballistic coefficient

$$B = \frac{C_d A}{m} \tag{4}$$

and where the parameters are given as:

B	Ballistic coefficient $(\frac{km^2}{kg})$
ρ	Atmospheric density $(\frac{kg}{km^3})$ { use either mean or maximum }
P	Orbital Period (min)
r	Distance from the Center of the Earth (km)

Table 2.1 Orbital Decay Table of Variables

From Equation (4), certain factors are apparent regarding the orbital lifetime of a satellite incapable of orbital transfers. The orbital decay rate is directly proportional to the drag coefficient (C_d), the presented area of the object (A), and the atmospheric density (ρ) and it is inversely proportional to the mass (m) and the orbital period (P), while being more sensitive to the orbital radius (r). Current estimates for the orbital lifetime of a nano-satellite in a 380 km orbit is 0.8 to 1.7 years. However, for the purpose of doing research in a dynamic experiment, a long lifetime is desirable. Since nano-satellites weigh on the order of 10 kg, the ballistic coefficient (B) is greater than that of an average satellite, and a decrease in the orbital decay rate is expected if no corresponding drop in $C_d \cdot A$ occurs. The desired goal is to place collaborative satellite experiments in higher orbits while exploring means to decrease downlink transmitter power. Other issues include atmospheric *fleet compaction* and atmospheric *fleet shearing*. With atmospheric fleet compaction, agents in the lead position of an orbiting fleet act as an aerosol shield. Satellites caught in this wake slowly approach leading satellites. Atmospheric fleet shearing is a result of two properties. First, satellites in higher orbits travel at lower velocities and are left behind.

Small Satellite Sub-Systems	
I	Mechanical Structure (bus)
II	Power Systems (solar, nuclear, etc.)
III	Telemetry (uplink/downlink)
IV	Attitude Control and Determination
V	Orbit Control and Determination
VI	Thermal Management and Control

Satellite Size	*Mass*
Large Satellite	> 1000 kg
Medium	500 - 1000 kg
Mini	100 - 500 kg
Micro	10-100 kg
Nano	1.0 - 10 kg
Pico	< 1.0 kg

Table 2.2 Small satellite sub-systems [left] and Satellite size classification [right]

Second, satellites in lower orbits experience more average atmospheric drag and; therefore, decelerate faster than their higher counterparts. Although orbital altitude and the atmospheric drag gradient produce a decrease in tangential velocity, shearing occurs and a constellation 'stretches' along the direction of travel over time.

2.5 Nano-Satellites

2.5.1 Overview

The United States space program currently faces a funding drought and has pursued a drive to decrease the size and power of both electrical and mechanical components. A result of the cutbacks is a range of devices with the same functionality (often better) as their macro functional equivalents. Tiny GPS receivers, tuning fork gyros, laser ring gyros, micro-mirror displays, LED displays, MEMS accelerometers, addressable nano-impulse thrusters and micro-motors already exist, with better designs in development. The Defense Advanced Research Projects Agency (DARPA) is providing incentives to both those who manufacture and purchase these devices. Thus satellites are getting smaller, are requiring less power, and gaining functionality at the same time. Nearly every sub-system on small satellites is affected.

The term 'nano-satellite' (Table 2.2) conjures images of a satellite far smaller than 1.0 kg to 10 kg, but when taken in the context of existing satellites, a 10kg satellite is indeed *nano* [40]. This thesis proposes a conceptual mission that is really *scale invariant,* but the term nano-satellite [2] appears more than once throughout the document as an aid to imagination. It is more difficult to imagine a swarm of 10,000 satellites, each 10 kilometers on a side, than a constellation of satellites 0.01 meters on a side. Regardless of scale, algorithms that facilitate meaningful collaborative behavior require further development.

2.5.2 The Resolution Solution

The question, 'Won't structures appear rough if made with cubes?' must be addressed. The resolution of a structure [4,33] made of identical components increases as the number of elements used to approximate it increases. For example, imagine a pixel approximation of the character 'a.' In Illustration (2), [left], we see a low resolution approximation of the letter 'a' while on the right we see a high resolution appriximation. If we were to divide the volume of the letter 'a' on the left by the number of identical pixel components, the ratio would be less than that in the example on the right. This same logic extends to three dimensions with orbital structures. A higher resolution structure is simply made of more components, much like a higher resolution computer screen. When passed through the human visual system, a higher resolution structure begins to look smooth rather than jagged. This illusion is due to the spatial frequency filtering of the human visual system. When designing a structure, metrics must be developed to select the resolution required to approximate the desired object. In the case of a parabolic dish, the resolution must be high compared to the received and/or transmitted wavelength.

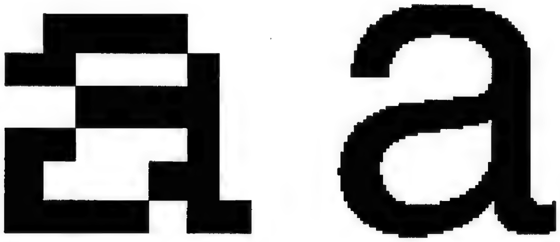


Figure 2.2 *Low Resolution and High Resolution Comparison*

3. Methodology

3.1 Introduction

A minimal set of satellite characteristics and one method for designing their behavior algorithms is presented. The agent [21,37], a term synonymous with nanosatellite, must be designed to perform optimally in the environment and for the given mission constraints. The structure formation mission imposes several interesting constraints, the first of which is similarity. Each agent must fit in a regular lattice and provide structural strength. This constraint reduces the list of 3D geometric shapes that can form solid lattices to structures such as tetrahedrons and hexahedrons (cubes) [4]. Like molecules in a crystal, satellites become members of a large lattice structure. The builder is converted to an element of the building in a process best termed 'assimilation.' The equilateral hexahedron (cube) is has the geometry of choice for several reasons. First, calculations are simplified because it easily aligns with a 3D Cartesian coordinate system. Second, it is symmetric about 12 unique planes, which implies a decrease in manufacturing costs. Also, 12 plane symmetry increases the probability of minimal rotation during satellite docking. A cube need only rotate a maximum of 45^o about an arbitrary vector to align for lattice assimilation. Tetrahedron and cylindrical honeycomb designs are more likely to require larger rotational maneuvers for pre docking alignment [2].

A behavior generation algorithm is developed that uses *apriori* information regarding structural satellite juxtaposition to convert global information to local action. Global information is passed in the form of single valued functions. The proposed *four-post* algorithm bridges the global-to-local gap, effectively reducing the probability of mission failure. To facilitate formation flying, reception transfer functions are developed that convert two channels of received light energy into satellite control level commands (each satellite has a tranceiver on every face). A flat field model is used to constrain transfer function coefficients and establish a realistic relationship between Transfer Function TF_0 for RX/TX channel-0 and TF_1 for channel-1.

3.2 The Agent

3.2.1 Introduction

Manufacturing costs and the probability of mission failure are both inversely proportional to simplicity. The STEMS satellite design is simplified by moving processor power [18,25,26] to the agent level and by using identical hardware for each satellite face. Only mission essential sensors and functionality exist in this design. Table (3.1) outlines four assumptions that increase simplicity.

Simplifying Assumptions	
Type	*Description*
Symmetry	Frequency selective transducers gauge light intensity for inter-sat dist. information.
Transmission	Structural agents communicate digital information via a simple metal contact.
Memory	Memory is only large enough to store the coefficients for one parametric equation.
Processor Power	Processor power is only large enough to evaluate the stored parametric equation.

Table 3.1 Simplifying assumptions

The following sections describe agent related concepts used in the STEMS model. These concepts are *thrust*, *rotation*, *translation*, *rotational inertia*, *body panel design*, and *the sensor package*. Assumptions made in the STEMS model are presented and validated.

3.2.2 Thrust

The age of MEMS technology [18] is now. Devices created on the MEMS level are generally smaller, less massive, and consume less power than their macro equivalents. Already, a number of proposed micro-impulse thrusters are in development using current technology. These new technologies are being leveraged by support from world governments, agencies such as the Defense Advanced Research Projects Agency (DARPA), and from the Air Force Research Laboratory Space Vehicles Directorate (AFRL/VSDD). Figure (3.1) illustrates one enabling technology important to future nano-satellites: a conceptual nano-propulsion system. In this system resistors below small propellant charges are addressed like memory. They generate

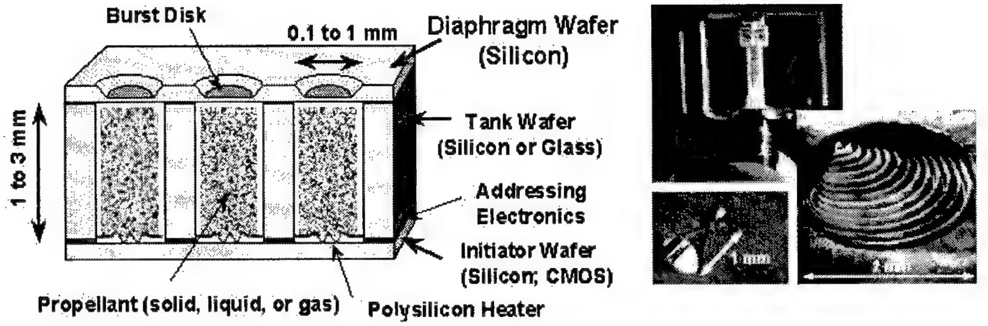

Burst Disk
0.1 to 1 mm
Diaphragm Wafer (Silicon)
Tank Wafer (Silicon or Glass)
Addressing Electronics
Initiator Wafer (Silicon; CMOS)
1 to 3 mm
Propellant (solid, liquid, or gas)
Polysilicon Heater
1 mm

Figure 3.1 MEMS micromachined thrustors for microsatellite propulsion [2,3]

enough heat when addressed to ignite a propellant charge, and a small explosion is set off. The result is a directed burst of thrust that equates to a specific change in velocity. Arrays of heating elements originally developed for infrared displays were modified to address and heat thousands of propellant containers to the point of explosion. Such enabling technologies make theoretical space science inventions easier to transition into functional systems.

One advantage of impulse thrusters is that they induce a near instantaneous change in velocity; the volume of propellant equates to a known impulse. Given both the magnitude of this impulse and the mass of a given satellite, the corresponding change in velocity is determined. The STEMS simulator takes this fact into account. Yet another advantage of impulse thrusters is MATLAB® code simplification. Acceleration does not need to be modeled if we couple both synchronous thruster firing and an impulse thrust model. The scene is analyzed at the beginning of each simulation frame and decisions are sent to virtual control systems on each of the N^c constellation satellites. To transition the current satellite velocity $(\overrightarrow{T^e})$ to the behavior function recommended velocity $(\overrightarrow{DS})$, the control system determines how many (n) impulse modules to address. After firing, the satellites coast along $\widehat{DS}$ with new velocities until the next constellation synchronous burn dt seconds later.

Synchronous firing further simplifies the STEMS model by reducing the number of simulation frames and by avoiding acceleration computation. Asynchronous systems are exceedingly time consuming to simulate, because the frame duration varies and a separate frame is required for every satellite. Acceleration changes are addressed in code because the impulse duration (ΔI) is considered small compared to the frame interval (dt). Figure (3.2) illustrates the real-world approximation of an ideal Dirac delta (δ) impulse acceleration.

Figure (3.2) illustrates the impulse approximation used in this thesis. Here, peak force Ip and thrust duration dI provide enough information to determine the effective impulse using

$$I = \int_{-\frac{dI}{2}}^{0}((t+\frac{\Delta I}{2})Ip)dt + \int_{0}^{\frac{dI}{2}}(Ip(1-t))dt \qquad (1)$$

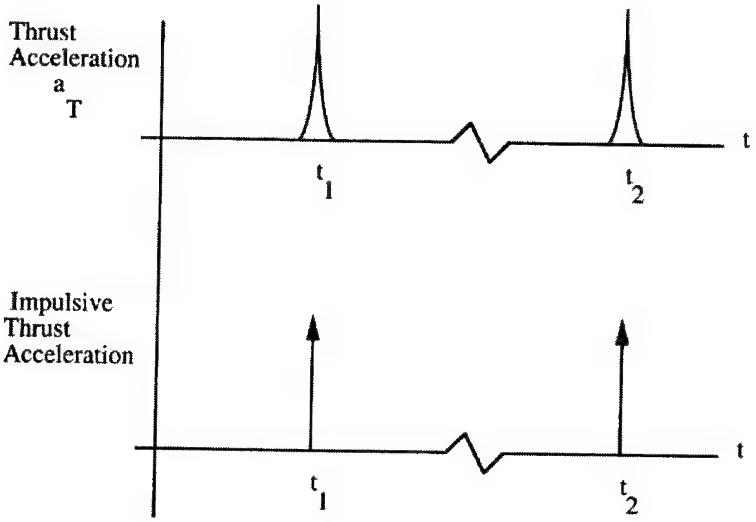


Figure 3.2 Thrust Acceleration (A_t) vs Time(s) *approximation allows for near instantaneous changes in velocity* [Prussing 35]

and simplifying this equation yields:

$$I = \left(\frac{\Delta I \cdot Ip}{2}\right) \frac{kg \cdot m}{\text{sec}} \tag{2}$$

The net change in velocity due to firing an impulse thruster(s), given the mass of an agent δ^{mass} in kilograms is then

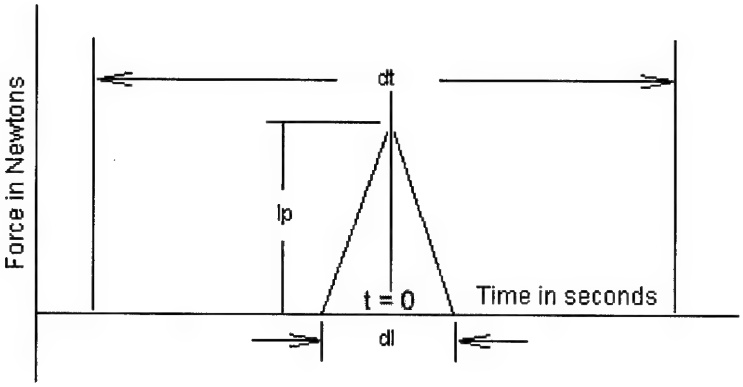


Figure 3.3 *Impulse thrustor approximation Force* (kgm/s/s) vs *Time*(s) diagram.

$$\Delta V = \left(\frac{\Delta I \cdot Ip}{2}\right) \frac{n}{\delta^{mass}} = \left(\frac{n \cdot I}{\delta^{mass}}\right) \quad \text{given that} \quad \Delta I \ll dt \tag{3}$$

in $(\frac{meters}{\text{sec}})$, where n is the number of simultaneous impulse firings, ΔI is the impulse duration, and I is the effective impulse of one thruster. STEMS stores ΔV figures in a matrix termed **IT** $(N^0 \times 3)$, which is stored in the data structure RUNN.F(I).IT. Agent mass (δ^{mass}) does not vary appreciably *between* satellites, because they are fabricated using the same process. Thus, δ^{mass} remains $10kg$ for every satellite in the STEMS model Again, we assume assembly line precision and consider that the CG of each satellite is centered precisely on the AR frame origin. Steps taken early on to simplify the design of a satellite can pay off in dramatic ways, yet it is possible to simplify a design too much so that the original goals cannot be realized.

3.2.3 Rotation and Translation

Since regular hexahedrons (cubes) are symmetric about 12 planes, deciding which direction is up is a formidable problem and one we never really have to solve, except to satisfy our own human desire to differentiate up from down. Thus, 'up' may be proclaimed the Z axis, which is perpendicular to both the X and Y axes. Furthermore, we align the X and Y axes with the sides of the satellite. Thus the origin of the AR Frame is placed squarely on the center of mass and at precisely the center of the satellite. If we adopt this convention, the control system and modeling code are greatly simplified.

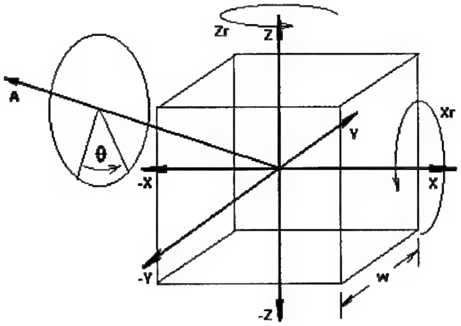


Figure 3.4 *Moments of rotation about the coordinate axes* (X,Y,Z) *and an arbitrary axis* (A)

3.2.4 Satellite Body Panels

The Structural Emergence Simulator (STEMS), coded in MATLAB®, provides a conceptual design for the body panel of an agent. Each face panel is identical to maintain even mass distribution and to decrease manufacturing costs. Each face must have bulkhead feed-throughs for power from solar panels. It must also have mounting positions for antennae and a two channel frequency selective line-of-sight transmitter/receiver transducer. Thrusters are positioned at each of the four corners because the distance from a cube center to the corners is $\frac{w\sqrt{3}}{2}$ and is maximal. Recall that $\tau = F \times d$, thusly, maximum torque is achieved when force is applied perpendicular to a maximal distance.

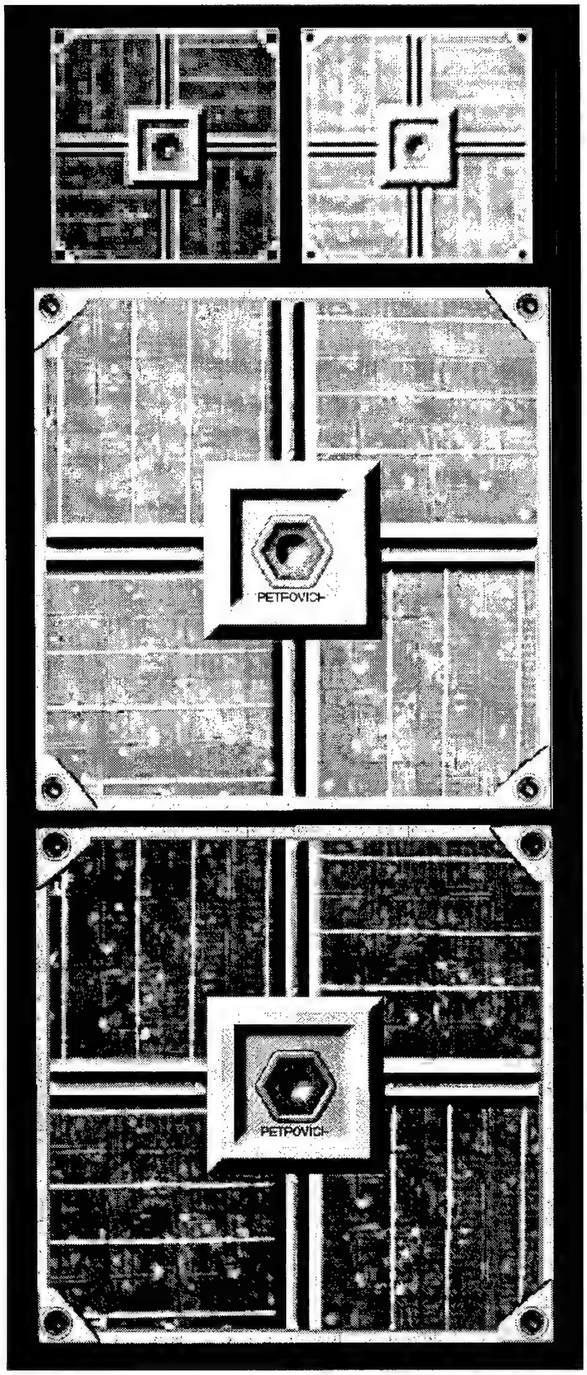


Figure 3.5 Color and Black/White (BW) Conceptual Satellite Faces: (upper left plate) *color; low resolution fast render*, (upper right plate) *BW, low resolution fast render;* (middle) *BW high resolution slow render;* (bottom) *Color high resolution slow render. Designed on December, 24, 1998.*

The spatial location of thrusters remains a function of the available range of thrust. In this configuration the satellite is capable of traveling faster along a vector that extends from corner point $(-\frac{w}{2},-\frac{w}{2},-\frac{w}{2})$ to $(+\frac{w}{2},+\frac{w}{2},+\frac{w}{2})$ of magnitude $\frac{w\sqrt{3}}{2}$. Each panel is symmetric about four axes, again, to simplify the design and distribute weight evenly. The optimal orientation for antennae is perpendicular to each face. In this orientation, the satellite face acts as a ground plane. However, this choice presents two problems. Antennae interfere with the directional light transducers and with face-to-face satellite docking. These problems are solved by using surface patch antennas that extend from the central sensor package radially outward.

3.2.5 Rotational Inertia

The STEMS model requires information on the rotational moments of both the structure and the constellation satellites. An equation is derived specifically for a satellite of overall width w and hollow center of width ε. A mass density variable (ρ) is added to further increase realism. A hollow satellite with an added mass density variable remains an approximation of the actual spatial distribution expected for a real satellite. The added complexity of batteries, thrusters, solar panels, wiring, etc., to the mass distribution is still taken into account if the variables ρ and ε are properly selected. The model approximates the mass distribution of a satellite with an empty shell with a uniform mass density ρ. We define the position matrix $\mathbf{P}$ of dimension $(N^s \times 3)$ and the structure center of gravity CG^s as

$$\mathbf{P} = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ & \vdots & \\ x_{N^s,1} & x_{N^s,2} & x_{N^s,3} \end{bmatrix} \quad \text{and} \quad CG^s = \begin{pmatrix} c_1 & c_2 & c_3 \end{pmatrix} \tag{4}$$

where 1, 2, and 3 correspond to x, y, and z, respectively. Extracting the columns of **P** yields

$$\mathbf{P}_x = \begin{bmatrix} x_{1,1} \\ x_{2,1} \\ \vdots \\ x_{N^s,1} \end{bmatrix} \quad \mathbf{P}_y = \begin{bmatrix} x_{1,2} \\ x_{2,2} \\ \vdots \\ x_{N^s,2} \end{bmatrix} \quad \mathbf{P}_z = \begin{bmatrix} x_{1,3} \\ x_{2,3} \\ \vdots \\ x_{N^s,3} \end{bmatrix} \tag{5}$$

We further describe a matrix $\mathbf{P}^{xy}$ that consists of two columns $\mathbf{P}_x$ and $\mathbf{P}_y$ perpendicular to the coordinate axis $\widehat{z}$, for example. We apply $\mathbf{P}^{xy}, \mathbf{P}^{xz}, \mathbf{P}^{yz}$ to find the moments of inertia $\mathbf{I}^x(\mathbf{P}^{xy}, w, \rho, \varepsilon)$, $\mathbf{I}^y(\mathbf{P}^{xz}, w, \rho, \varepsilon)$, $\mathbf{I}^z(\mathbf{P}^{yz}, w, \rho, \varepsilon)$, respectively and a root radius $\mathbf{R}_c$ is defined to simplify the calculations. Given some $\mathbf{P}_a$ and $\mathbf{P}_b$

$$\mathbf{R}^c = \left(\mathbf{P}_a^2 \;\; \mathbf{P}_b^2 \right), \text{ where } (a \neq b, a \neq c, b \neq c) \text{ and } \{a, b, c\} \;\epsilon\; \{\widehat{\text{x}}, \widehat{\text{y}}, \widehat{\text{z}}\} \tag{6}$$

Here $\{c\}$ is considered the coordinate axis of interest and each element of **P** is squared. Figure (3.6) illustrates the method for determining the moment of inertia about any given point *and* about some radial arm of length R'=$\sqrt{\mathbf{R}^c}$. For example, by integrating over infinitesimal volume elements and multiplying by the homogenous material density, we find the moment of rotational inertia ($\overrightarrow{I^c}$). The inertia about $\overrightarrow{I^c}$ for a three dimensional hollow cube is

$$\begin{aligned} I_j^c(\mathbf{P}_j^{ab}, w, \rho, \varepsilon) &= \int_{c'-\frac{w}{2}}^{c'+\frac{w}{2}} \int_{b'-\frac{w}{2}}^{b'+\frac{w}{2}} \int_{a'-\frac{w}{2}}^{a'+\frac{w}{2}} ([x_{j,a}^2 + x_{j,b}^2]\rho)\; da\; db\; dc \qquad (7) \\ &- \int_{c'-\frac{w}{2}}^{c'+\frac{w}{2}} \int_{b'-\frac{w}{2}}^{b'+\frac{w}{2}} \int_{a'-\frac{w}{2}}^{a'+\frac{w}{2}} ([x_{j,a}^2 + x_{j,b}^2]\rho)\; da\; db\; dc \\ & \qquad (8) \end{aligned}$$

Here ε delineates the width, depth, and height of the *missing* internal cube, $0 \leq \varepsilon < w$. Furthermore, j refers to the jth row of any given matrix and $\{a, b\} \epsilon \{x, y, z\}$. Given that $\mathbf{P}_j^{ab} = [\mathbf{P}_{j,a}, \mathbf{P}_{j,b}]$, Equation (7) simplifies to

$$I_j^c(\mathbf{P}_j^{ab}, w, \rho, \varepsilon) = \rho(w^3 - \varepsilon^3)\mathbf{R}_j^c + \frac{\rho}{6}(w^5 - \varepsilon^5) \tag{9}$$

Equation (9) can be used to find the moments of rotation about selected coordinate axis for a complex structure using

$$I^c = \rho(w^3 - \varepsilon^3) \sum_{j=0}^{N^s-1} \mathbf{R}_j^c + \frac{\rho\, N^s}{6}(w^5 - \varepsilon^5) \tag{10}$$

where $c : c \in \{x, y, z\}$ remains an arbitrary coordinate axis. We compute the moment of rotational inertia for some arbitrary satellite in a large structure, i.e., some large R_1', and compare it to the $\overrightarrow{I^z}$ of agent δ_0^s centered at the origin ($\mathrm{R}_2' = 0$). Example: select $\rho = 15.2207 \, \frac{kg}{m^3}$, $w = 1.0$ *meters*, $\varepsilon = 0.7$ *meters*, # $P^{xy} = [15.5m, 15.5m]$ and $R^z = [240.25m^2, 240.25m^2]$ for a satellite far from the origin. Substituting into Equation 10, we find $\overrightarrow{I^z} = 4{,}807.1 \; kg \cdot m^2$ at a distance of $R_1' = 21.92$ meters

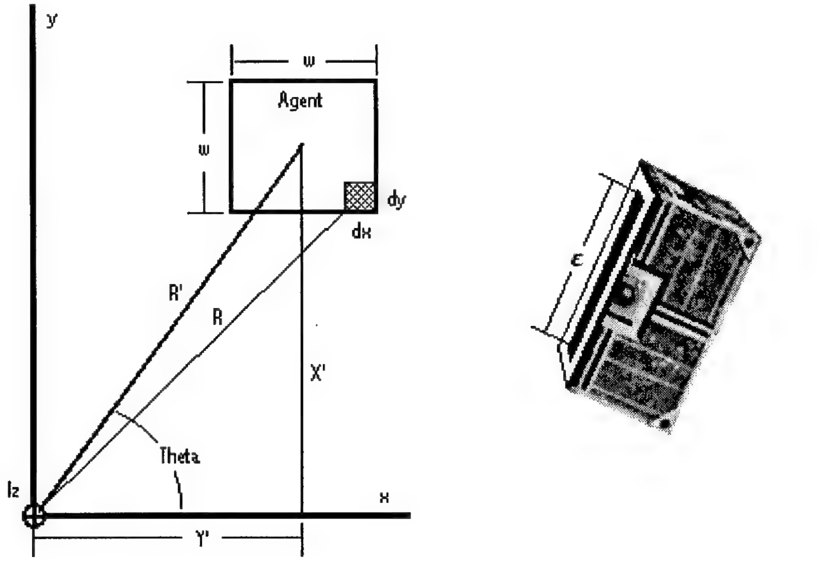


Figure 3.6 Calculating the moments of inertia for a single satellite.

from the origin. If $R_2' = 0$, then $\overrightarrow{I^2}$ reduces to $\frac{\rho N^s}{6}(w^5 - \varepsilon^5)$ and equals 2.1104 $kg \cdot m^2$ for the same values of w, ρ,and ε.

3.2.6 Sensors and Channels

Two channel photo-transmitters broadcast at 100 mWatts of power in a direction normal to each of the six satellite faces. Channel wavelengths are pre-selected based on noise levels in the environment. For example, if noise is minimal near a 880 nm wavelength, then this wavelength should be used for a channel. Receiver wavelengths are separated by a sufficiently large guard band and are filtered to minimize cross signal correlation. Also, the signal transmitter is not modeled as an isotropic radiator; instead, it is modeled as a directional transmitter. Off-angle transmission power decreases to a fraction $\cos(\theta)$ of the incident radiation, where θ is the angle between a normal sensor vector and a vector from the transmitter to a distant receiver as shown in Figure (3.7) For example, less power is radiated 30 degrees off normal than 10 degrees off normal. This model attracts satellites positioned along off-angles (θ_1: $\theta_1 < \theta_2$) with a greater velocity. In the STEMS code, gain variables 'K_XMIT' and 'K_REC' (k_1, k_2) are considered to be the same for every satellite in the constellation. Recall that δ_i^p is the position of a satellite/agent (i) relative to the structural reference

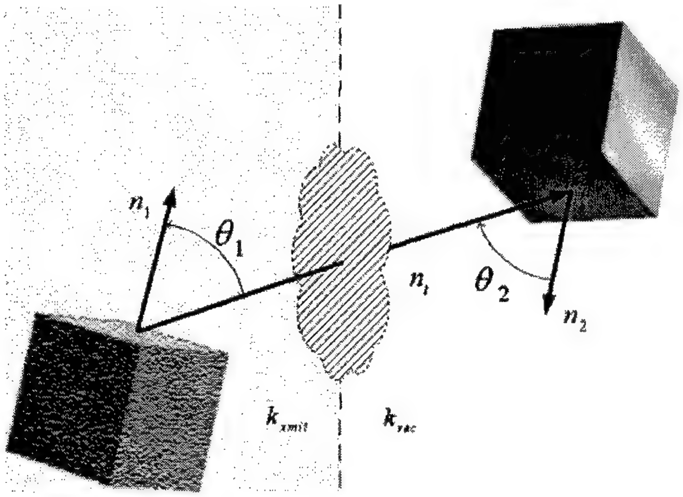


Figure 3.7 Off-nadir optical duplex tranmission model

frame. The directional nature of the transducers is taken into consideration using

$$m_{i,2}(k_1, k_2, \delta_1^p, \delta_2^p, \overrightarrow{n}_1, \overrightarrow{n}_2) = \frac{m_{i,1}, k_1, k_2, (\overrightarrow{n}_t \bullet \overrightarrow{n}_1)(-\overrightarrow{n}_t \bullet \overrightarrow{n}_2)}{d^2} \tag{11}$$

where m_1, the radiated power from one transducer of $\delta_1$is

$$m_{i,1} = \| \overrightarrow{n}_1 \|, \tag{12}$$

where $\overrightarrow{n}_t$, a vector pointing from RX (δ_1^p) to TX (δ_2^p), is

$$\overrightarrow{n}_t = (\delta_1^p - \delta_2^p) \tag{13}$$

and d, the transmitter-to-receiver distance, is given in terms of $\overrightarrow{n}_t$ by

$$d = \| \overrightarrow{n}_t \| \tag{14}$$

and finally k_1 and k_2 are the receiver and transmit gain variables, respectively. Calculating the received power on any single receiver is a computationally intensive task. It is functionally equivalent to a ray tracing algorithm, except that no intersection calculations are required. A summation of the received light from every other satellite in the constellation is required to determine the net received power on a given facial sensor, so the number of computations has a polynomial dependance on the number of swarming satellites.

Shadowing conditions are modeled using the mean shadow approximation developed in *Appendix A*. The shadow cast by a cube is approximated with the shadow cast by a sphere of width $\frac{w\sqrt{3}}{2}$ using the mean shadow approximation. Net received power for a satellite (i) for face (p) on channel-0 is

$$RP^0_{i,p} = \sum_{j=0}^{N^s-1} \sum_{q=1}^{6} m_{i,0}(k\_{xmit}, k\_{rec}, \delta^p_i, \delta^p_j, \overrightarrow{n}_{i,p}, \overrightarrow{n}_{j,q}) \qquad (15)$$

and the expression for channel-1 is determined by

$$RP^1_{i,p} = \sum_{j=0}^{N^s-1} \sum_{q=1}^{6} m_{i,1}(k\_{xmit}, k\_{rec}, \delta^p_i, \delta^p_j, \overrightarrow{n}_{i,p}, \overrightarrow{n}_{j,q}) \qquad (16)$$

given that $m_{i,p}(k\_{xmit}, k\_{rec}, \delta^p_i, \delta^p_j, \overrightarrow{n}_{i,p}, \overrightarrow{n}_{j,q}) > 0$ and that δ_i is indeed transmitting on the proper channel (0, 1, or 2) as indicated by the Emitter channel Matrix (**ECM**). If channel-2 (no transmission) is selected, then no power is added to the sum. If channel-1 is transmitting and incident, i.e., not shadowed, then it is added to $RP^1_{i,p}$.

The same is done for channel-0. A minimum of $(6\ N^s)^2$ repeated computations and Boolean expressions are required to update the $(N^s \times 6)$ received power matrices $RP^0_{i,p}$, and $RP^1_{i,p}$ for every satellite in the constellation. This added realism produces a final result commensurate with existing hardware..

3.3 The STEMS Agent Model

3.3.1 Data Storage Structures

The MATLAB® Structural Emergence Simulator (STEMS) was written to demonstrate the feasibility of nano-satellite structure formation. It generates two file types for massive data storage using the proprietary MATLAB® '*.mat' format. Files are automatically renamed with STEMS to include a '_i.mat' or '_r.mat' extension. Each file contains one of two data structures, AGENT and RUNN (Table 3.2). For example, files named 'simulation1.mat' become 'simulation1_r.mat', for a simulation run. The corresponding initial conditions file is changed from 'initial_ config .mat' to 'initial_ config_i .mat'. Each of these files contains a data structure.

Struct	*Child*	*Subs*	*Description*
RUNN	⇒	dt	(scalar) stores the real-world time between synchronous burns
		xp	Transmit power in milli-Watts
	f	**P**	$(N^0 \times 3)$ position matrix stored once per frame
		T	$(N^0 \times 3)$ velocity vector matrix
		R	(N^0x3) arbitrary rotational momentum vector matrix
		B	$(8 \times 3 \times N^0)$ body frame paged matrix
		S	$(6 \times 3 \times N^0)$ normal sensor vector matrix
		RP0	$(1 \times 6 \times N^0)$ {+X,-X,+Y,-Y,+Z,-Z} received pwr matrix (Channel 0)
		RP1	$(1 \times 6 \times N^0)$ {+X,-X,+Y,-Y,+Z,-Z} received pwr matrix (Channel 1)
		RP2	$(1 \times 6 \times N^0)$ {+X,-X,+Y,-Y,+Z,-Z} received pwr matrix (Channel 2)
		ECM	$(1 \times 6 \times N^0)$ Emitter Channel Matrix:. each element is either 0,1,or 2
		IT	$(N^0 \times 3)$ thrust vector matrix.
		IR	$(N^0 \times 3)$ rotational momentum matrix.
		stm	$(1 \times N^0)$ structural membership (**stm**) matrix. binary 0 = no 1 = yes
AGENT	*phys*	p	$(N^0 \times 3)$ position relative to the origin (meters)
		t	$(N^0 \times 3)$ translation velocity (meters/sec)
		r	$(N^0 \times 3)$ moment of rotation
		m	(scalar) mass (kg)
		i	(scalar) rotational inertia (kgm^2)
		b	$(8 \times 3 \times N^0)$ inital body frame orientation
		s	$(6 \times 3 \times N^0)$ initial six orthogonal sensor vectors
		trans	(structure) transmission structure [XP,RP]
	disp	color	(1×3) reder mode 0 solid satellite color
		vert	(8×3) a square set of corner points on a cube
		tvert	$(6\ x3)$ a square set of normal sensor vectors
		imap	$(M\ x\ M)$ image map for mode 1 satellite rendering
		cmap	$(M\ x\ 3)$ colormap for the image map
		dbit	$(scalar)$ selects the rendering mode (0,1,2)
		fac	(6×4) connection sequence for **vert** body frame matrix

Table 3.2 Data storage diagram for the AGENT and RUNN data structures

The first data structure, associated with the '_i.mat' filename, is termed AGENT. The second is associated with the '_r.mat' filename and is termed RUNN. Data structures allow a user to access submatrices of a parent structure using a dot $(\cdot)$ operator. For example, to access the position matrix **P** in the AGENT data structure, one types **agent.phys.p** at the command prompt, assuming that AGENT is resident in memory.

Underscore 'r.mat' files store simulations as a sequence of frames separated by the time interval *dt* (seconds). The runn structure in this underscore 'r.mat' file can be prohibitively large for 300 frame simulations of 400+ satellites. However, data compression ratios are generally in the 50% range for both '_i.mat' and '_r.mat' files. In addition, they are easily ported via FTP between computers.

Table (3.2) shows the initial ephemerides stored and accessed by the Generate Initial Conditions File (GICF) editor. File sizes for '_i.mat' initial conditions files are considerably smaller in size than their '_r.mat' counterparts. They are equivalent in size without the potentially large image files, used for satellite texture mapping, stored in the *imap* field.

3.3.2 Body Matrix

The (8 x 3) body matrix **B** stores the corner positions of a cube. The **vert** field of the agent structure holds an initial wire-frame model of a satellite. If we view **vert** before it is stored in **B** and rotated from frame to frame arbitrarily, it is:

$$\mathbf{vert} = \begin{bmatrix} -0.5 & -0.5 & +0.5 \\ -0.5 & -0.5 & -0.5 \\ +0.5 & -0.5 & -0.5 \\ +0.5 & -0.5 & +0.5 \\ -0.5 & +0.5 & +0.5 \\ -0.5 & +0.5 & -0.5 \\ +0.5 & +0.5 & -0.5 \\ +0.5 & +0.5 & +0.5 \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} -0.6373 & -0.1836 & +0.5569 \\ -0.4099 & -0.7183 & -0.2569 \\ +0.5609 & -0.6594 & -0.0244 \\ +0.3335 & -0.1246 & +0.7895 \\ -0.5609 & +0.6594 & +0.0244 \\ -0.3335 & +0.1246 & -0.7895 \\ +0.6373 & +0.1836 & -0.5569 \\ +0.4099 & +0.7183 & +0.2569 \end{bmatrix}$$

Here **B** is actual data taken from the 23rd frame of a simulation. It contains eight vectors that describe the eight corners of a rotated cube. We know that this cube was rotating slowly, because after 23 frames the signs on each element remain identical to the aligned model **vert**. To render the sides of the cube as four corner patches, the **fac** matrix is used. The **fac** matrix addresses the rows in **B**, which correspond to points in three dimensions. MATLAB$^{\circledR}$ is capable of rendering N cornered patches in three dimensions and can also texture map the region inside these patches or simply apply a flat color specified by *agent.disp.color.*

$$\mathbf{fac} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 6 & 7 & 3 \\ 4 & 3 & 7 & 8 \\ 1 & 5 & 8 & 4 \\ 1 & 2 & 6 & 5 \\ 5 & 6 & 7 & 8 \end{bmatrix}$$

Body frames are stored for each satellite in the constellation in three dimensional matrices. The precision of MATLAB$^{\circledR}$ is addressed in *Appendix A.* In body frame modeling, if the same body frame is manipulated over a large number of frames, then error accumulates and the body frame becomes mal-aligned. From a computational standpoint, using the same body frame repeatedly is much faster. The result of the error analysis is that a body frame may be *repeatedly rotated and translated* over tens of thousands of iterations before a one part in one-trillion error accumulates.

3.3.2.1 Ephemerides. Three critical ephemerides are; position, velocity, and rotation. The position matrix **P** gives the positions of the centers of gravity for each agent. The center of gravity is considered the point (0,0,0) relative to the body matrix (**B**) and sensor matrix (**S**). In STEMS, translation is accomplished by adding the elements of **P** to the columns of **B** $(\mathrm{B}(:,:,i) = B(:,:,i-1) + ones(8,1) \cdot P(i-1,:))$

$$\mathbf{P} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} \\ p_{2,1} & p_{2,2} & p_{2,3} \\ & \vdots & \\ p_{N^o,1} & p_{N^o,2} & p_{N^o,3} \end{bmatrix} \qquad \mathbf{T} = \begin{bmatrix} t_{1,1} & t_{1,2} & t_{1,3} \\ t_{2,1} & t_{2,2} & t_{2,3} \\ & \vdots & \\ t_{N^s,1} & t_{N^s,2} & t_{N^s,3} \end{bmatrix}$$

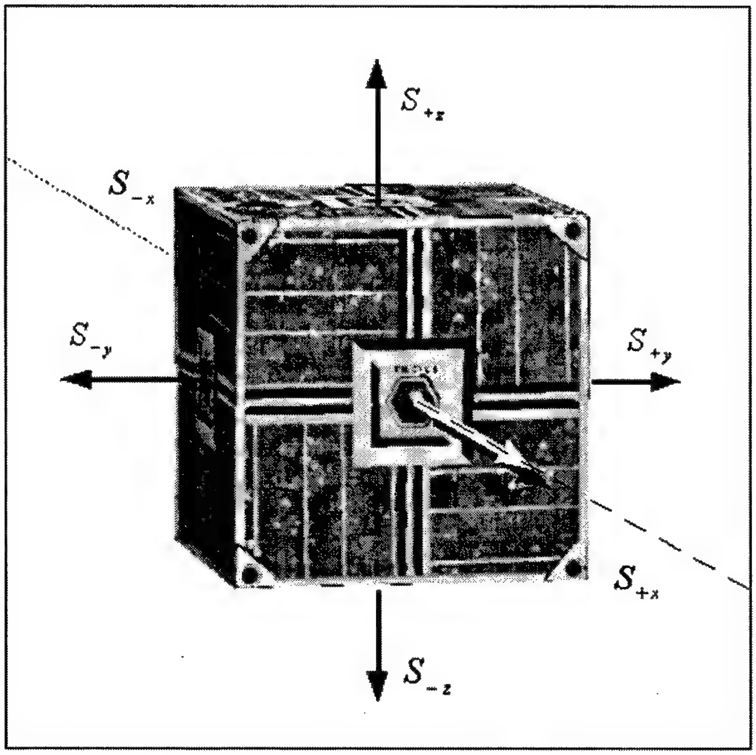


Figure 3.8 Maximum gain vectors normal to the six on-board 3-channel photo-transmitter/receiver modules

The velocity matrix (**T**) is also relative to the center of gravity of an agent. It provides the current direction and velocity of the satellite at any given time. The magnitude of this velocity vector is satellite speed.

Lastly, the moment of rotation vector (**R**) is relative to the satellite position (**P**) and provides two pieces of important information. First, the magnitude of **R** is the angular velocity. Second, the direction of **R** defines the axis of rotation and direction of rotation obeys the right hand rule.

$$\mathbf{R} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ & \vdots & \\ r_{N^s,1} & r_{N^s,2} & r_{N^s,3} \end{bmatrix}$$

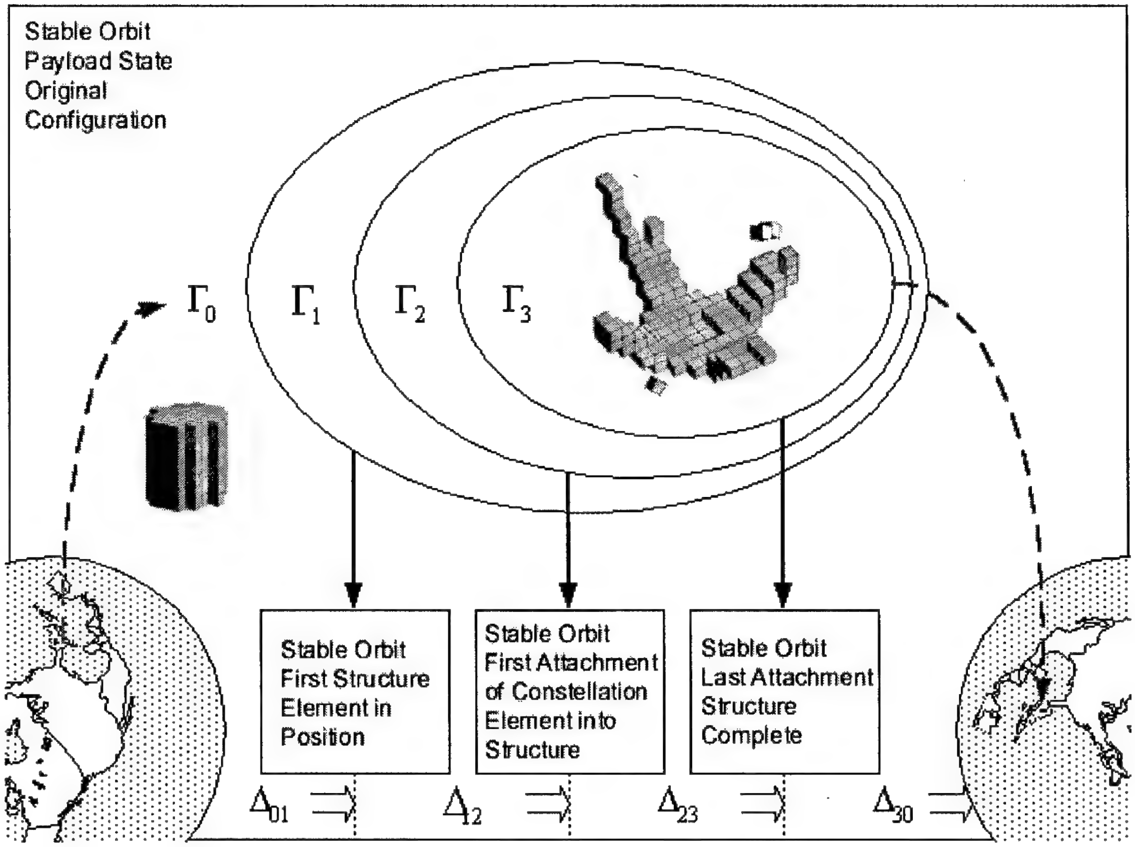


Figure 3.9 Four primary phase transitions: *from initial configuration to re-configuration.*

3.3.2.2 Structural Membership Matrix. The structural membership matrix (**stm**) is stored in *runn.f(j).stm* throughout a simulation and is a (1 x N^0) matrix filled with binary 0*s* or 1*s* that correspond to individual satellites. Here, $\mathbf{stm}_{1,3}$ refers to satellite 3 in the constellation.

$$\mathbf{stm} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \qquad (17)$$

If $\mathbf{stm}_{1,3}$ ==1, then we know that δ_3^c is part of the structure and a member of β^s. If **stm**(1,3)==0, then we know that δ_3^c is still a member of β^c. Furthermore, we assume that no satellite is ever eliminated from the constellation. A third state is added to the **stm** matrix to provide for this possibility.

3.4 Behavior Generation

3.4.1 Behavior Transfer Function

Hereafter, the transfer function that facilitates swarming [6,8,22] is referred to as the *binary* method. The term 'binary' is fitting because it indicates that two light intensity communications channels (0 and 1) are used. Later, the 'four-post' face activation algorithm is defined. Together the *binary* method and the *four-post* algorithm accomplish the complex structure formation mission with a success rate greater than expected given the simplicity of the conceptual satellites. Equations (15), (16), and a current velocity estimate ($\overrightarrow{T^{e}}$) provide all of the local information required to accomplish a *reasonable* structure formation mission. However, to facilitate formation flying a more precise version of swarming that uses exact position information is required. No precise position information is allowed with the *binary* method because it relies too heavily on other systems; namely, GPS. Should the GPS signal(s) be jammed or spoofed, then the entire constellation is rendered inoperative. In addition, one can envision a formation working in an environment far from earth and

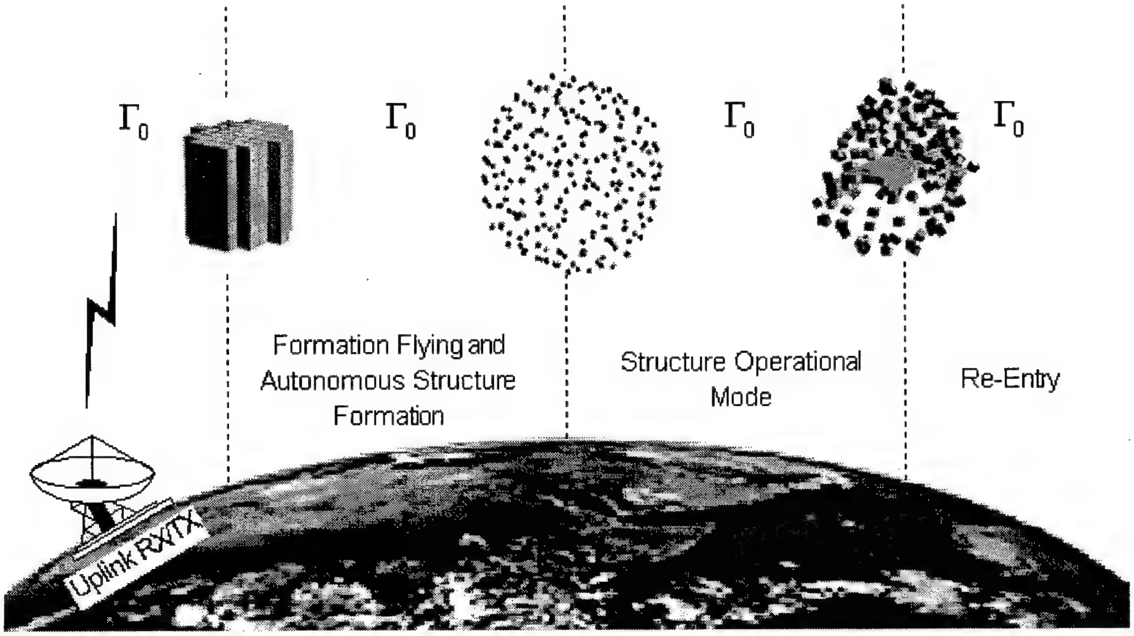

Γ_0
Γ_0
Γ_0
Γ_0
Formation Flying and Autonomous Structure Formation
Structure Operational Mode
Re-Entry
Uplink RX/TX

Figure 3.10 Orbital mission: *mission requirements are sent via uplink to the autonomous constellation.*

GPS. An alternative is precise relative positioning, where again, added complexity in the form of a digital inter-satellite communication system is required for formation flying. Swarming, or imprecise stochastic cellular automata [21,22,34,35] style behavior, does not require such added complexity. However, even the *binary* method requires an agent-level intelligence sufficient to compute the solutions to simple vector equations.

As an example, consider one satellite (δ^c_{1003}) as it traverses a path throughout a swarm. This satellite samples and updates on-board $\mathbf{RP}^0_{1003}$ and $\mathbf{RP}^1_{1003}$ registers shortly before firing impulse thrusters. Agent 1003 then calculates a current velocity estimate ($\overrightarrow{T^e}$) based on the time varying differences in $\mathbf{RP}^0_{1003}$ and $\mathbf{RP}^1_{1003}$, where the matrix T^e is relative to the agent reference frame (AR_{1003}). The satellite then computes

$$\overrightarrow{W^0_{j,:}} = \mathbf{W}^0_{j,:} = \mathbf{RP}^0_{j,:} \cdot \mathbf{S} = \mathbf{RP}^0_{j,:} \begin{bmatrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad (1 \le j \le N^c) \qquad (18)$$

for the net reception on channel-1. Similarly, it calculates $\overrightarrow{W^1_j} = \mathbf{RP}^1_{j,:} \cdot \mathbf{S}$, where $\mathbf{S}$ is a common sensor vector matrix (Figure 3.8) and where the general satellite *row* index j is instead of j=1003 for this example. Recall that channel-0 facilitates social swarming behavior whereas channel-1 induces structure formation by providing a strong attractive force to particular faces. The (1 × 3) vectors $\overrightarrow{W^0_j}$ and $\overrightarrow{W^1_j}$ are termed 'power vectors'; they point in the direction of greatest received power. The magnitudes $\|W^0_j\|$ and $\|W^1_j\|$ are indicators of the net received power from the $\widehat{W^0_j}$ direction.

In STEMS, δ^c_j calls a function $[\overrightarrow{DS^0_j}] = p\_field(\|W^0_j\|, \widehat{W^0_j}, mood0).m$. The magnitude of $\overrightarrow{W^0_j}$, a normalized version of $\overrightarrow{W^0_j}$, and a data structure termed *mood0* are passed to *p_field*. The data structures *mood0* and *mood1* are the two most significant constants in STEMS. Together, they pass constants to the *p_field* behavior transfer function, which converts them to thruster commands. The balance of social and structural forces must be carefully selected to avoid ill-conditioned behavior. The *p_field* transfer function TF^0 calculates

$$\begin{aligned} k &= mood0.k \quad \text{(scalar)} \\ A &= mood0.A \quad \text{(scalar)} \\ \overrightarrow{DS_j^0} &= -\widehat{W_j^0}\cdot\left(\left(\frac{A(\|W_j^0\|-k)}{(\|W_j^0\|+k)}\right)\cdot\begin{bmatrix}1 & 1 & 1\end{bmatrix}\right) \end{aligned} \tag{19}$$

where the vector $\overrightarrow{DS_j^0}$ is a 'desired direction' vector of dimension (1 x 3), and DS^1 is calculated in the same manner except that the structure *mood0* is passed to *p_field* instead of *mood1.*

The STEMS *behave1_b.m* function calls both $[DS_j^0] = p\_field(\left\|\overrightarrow{W_j^0}\right\|, \widehat{\overrightarrow{W_j^0}}, mood0).m$ and $[DS_j^1] = p\_field(\left\|\overrightarrow{W_j^1}\right\|, \widehat{\overrightarrow{W_j^1}}, mood1).m$ when required. After calculation, two desired directions exist. One desired direction is optimal from a social standpoint and the other is optimal from a structure formation standpoint. A compromise between $\overrightarrow{DS_j^0}$ and $\overrightarrow{DS_j^1}$ must be reached to avoid collisions and to facilitate swarming. The compromise equation is simply:

$$\overrightarrow{DS} = \left(\overrightarrow{DS_j^0}+\overrightarrow{DS_j^1}\right)\ \frac{meters}{\mathrm{sec}} \tag{20}$$

If the satellite is in danger of striking a nearby satellite, then $\overrightarrow{DS_j^0}$ may outweigh $\overrightarrow{DS_j^1}$ and the satellite will tend toward $\overrightarrow{DS_j^0}$. The direction of travel is ultimately determined by equation (3.19) rather than by equation (3.20). The problem with the latter equation is that it often requests action outside the physical capabilities of a real system. Consider the following scenario; a satellite is traveling with velocity $\overrightarrow{T^{e}} = [1,0,0]$ along the X-axis.

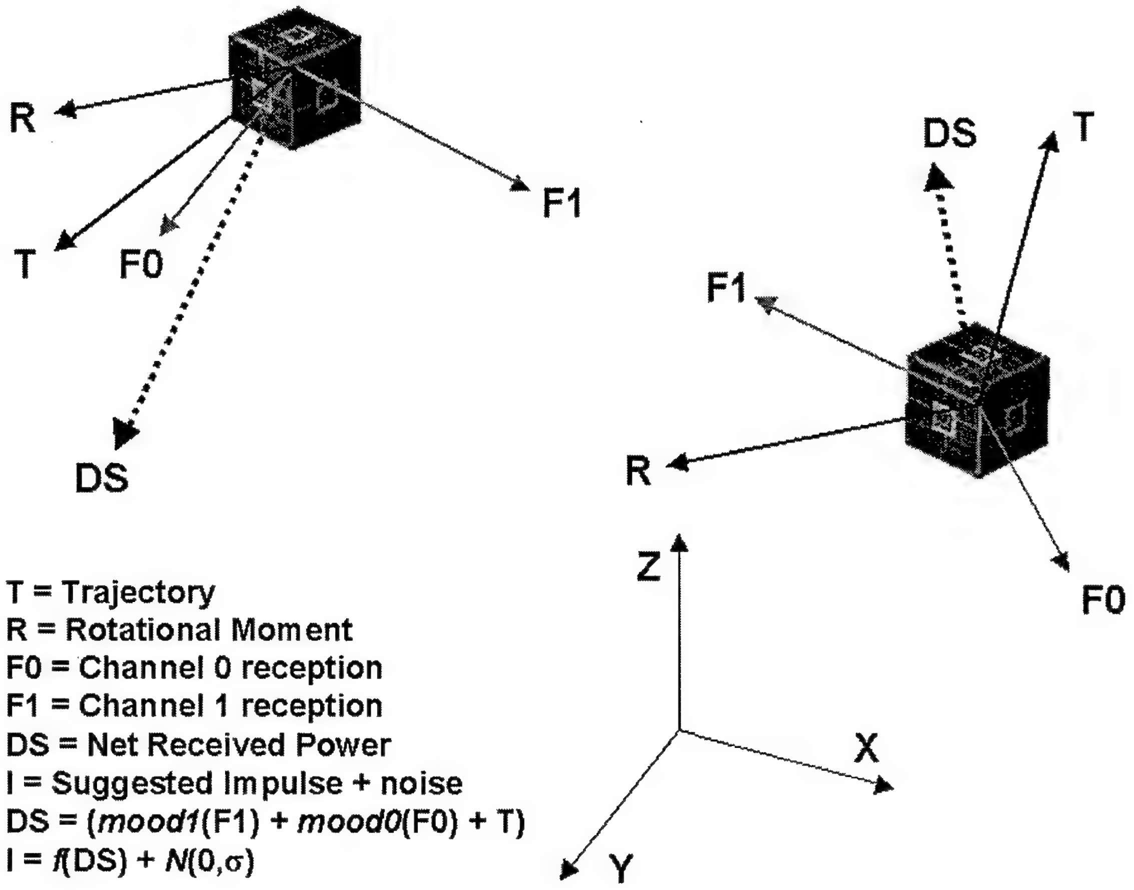


Figure 3.11 *Vector mechanics and the STEMS behavior1_b.m function.* Net received power from both channels is vectorized and passed through a transfer function. The binary behavior transfer functions for channels 0 and 1 convert the received power vector into a desired direction vector, which the control system attempts to realize [Beer 5].

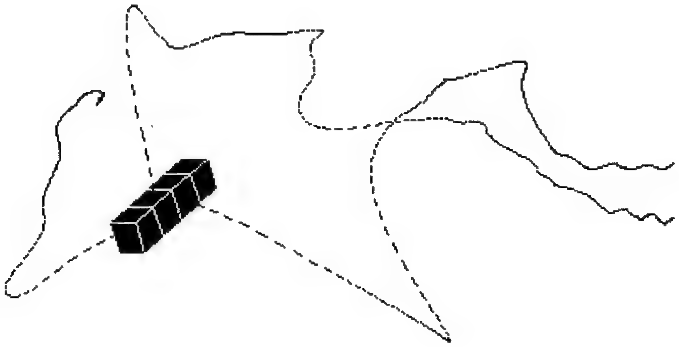


Figure 3.12 Three of four satellites seek equilibrium about the first with zero additive system noise. The two right-most satellites exhibit pairing behavior. One satellite is left fixed at the origin, while three are set free to swarm. One satellite decides to travel left, while two opt to travel right. Ultimately, they reach a state of oscillating, paired equilibrium due to the binary behavior algorithm.

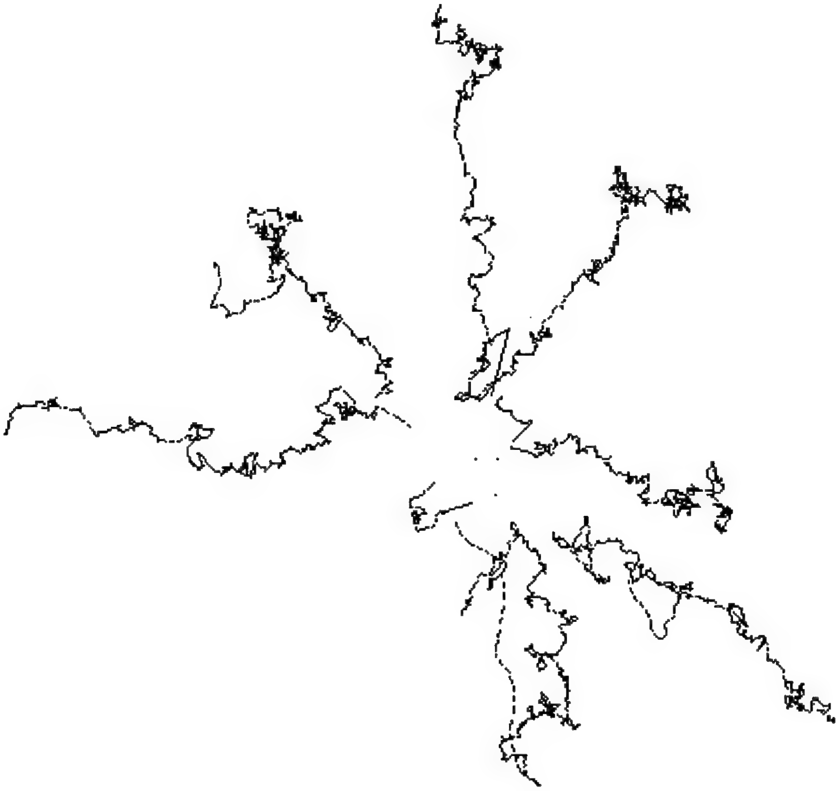


Figure 3.13 Ten satellites seek relative equilibrium positions with additive system noise by moving radially outward from a dense payload configuration. The annealing effect is apparent in the path taken at 120 degrees. Notice the dense cluster in the middle of this path. Such dense clusters form when satellites reach a point of near equilibrium. This satellite successfully broke free and continued on, thus proving that noise induces and annealing effect.

External conditions cast an incident $\mathbf{RP}_j^0$ and $\mathbf{RP}_j^1$ that ultimately yields $\overrightarrow{DS} = [0, 0, 1]$. No realistic thruster firing sequence is capable of sending a satellite instantaneously from direction $\overrightarrow{T^{e}}$ to $\overrightarrow{DS}$ (an instantaneous change in velocity of 90 degrees). Since thruster firings are synchronous and separated temporally by an interval dt, several (or more) subsequent thruster firings are required to complete the turn. STEMS results with the path tracking mode on demonstrate graceful arcing maneuvers (Figure 3.12), whereas paths taken with additive system noise (Figure 3.13) appear jagged.

In effect, a simple behavior equation (Equation 19) implements long complex maneuvers by indirectly decomposing the problem into a series of smaller maneuvers. One other reality constraint imposed on the transition from $\overrightarrow{T^{e}}$ to $\overrightarrow{DS}$ is the available thruster power. A linear range of thruster power is assumed to be between 0.0 and 10.0 N-s. Such a burn can send a 10 kg satellite from 0.0 to 1.0 m/s in ΔI seconds. However, an analysis (Chapter 6) indicates that the minimum thruster power be set at a level determined by the system noise.

Experimental results demonstrate that much less maximum impulse [1,5] is actually required. Burn-to-burn time (dt) is inversely proportional to the maximum available impulse, i.e., more lower magnitude bursts over the time period T is equivalent to fewer (less n) intense burns during the same interval (T).

Finally, a single vector is sent to the thruster control system

$$\overrightarrow{IT} = \left\{ \begin{array}{lr} 0 & \left(DS - \overrightarrow{T^{e}}\right) < I_{\min} \\ \left(DS - \overrightarrow{T^{e}}\right) & I_{\min} < \left(DS - \overrightarrow{T^{e}}\right) < I_{\max} \\ I_{\max}\widehat{\left(DS - \overrightarrow{T^{e}}\right)} & I_{\max} < \left(DS - \overrightarrow{T^{e}}\right) \end{array} \right\} \qquad (21)$$

and the components are converted to thrust. Recall that the thrust equation is

$$\Delta V = \frac{n \cdot I}{\delta^{mass}} \qquad (22)$$

Thus thrust magnitude increases as $n : \{\, n \,\epsilon\, \mathcal{I}\}$, the number of burst disks addressed simultaneously.

3.4.2 Repulsion and Attraction

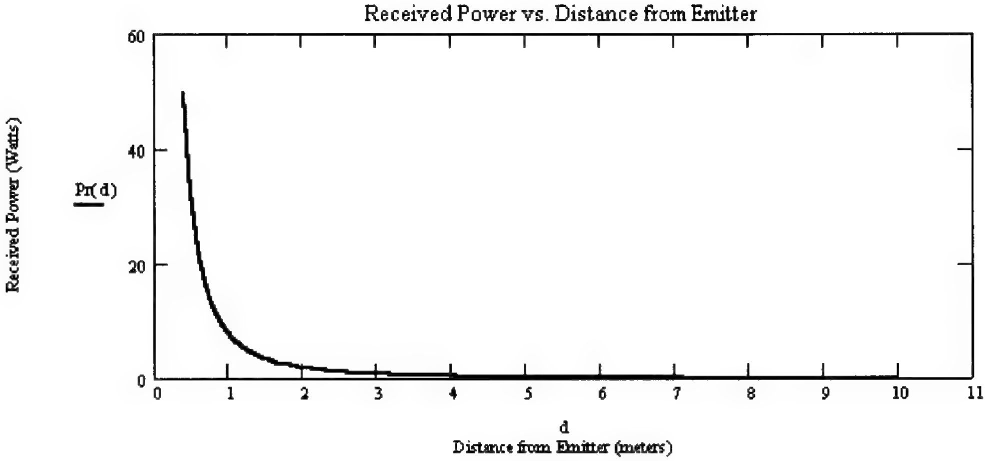


Figure 3.14 Received power from one emitter as a function of distance.

Figure (3.14) illustrates received power as a function of distance for a single transmitter-receiver pair. Power received from distant transmitters is low compared to power from near transmitters. Received power diminishes with distance according to the Friis Transmission equation:

$$P_r(d) = \frac{k\_rec \cdot k\_xmit \cdot P_d}{4 \cdot \pi \cdot d^2} \tag{23}$$

This function is shifted in STEMS so that the maximum received power is 100 mW. A maximum occurs when one transmitter is directly coupled to a different receiver in the payload or structural state.

The equilibrium position between an aligned single transmitter/receiver pair is where F1(d) + F0(d) = 0 or where the attractive and repulsive forces induced by receive power from channels 0 and 1 are equal. However, the equilibrium distance for a 2-satellite constellation is much different than the distance in an N^0 satellite constellation, because the sum of multiple transmissions causes equilibrium distances to increase.

Equation (19) (the behavior equation) is a variant of the Repulsive response vs. Distance equation given by the expression:

$$k_0 = \sqrt{\frac{P_t}{4\pi de_0^2}} \text{ given } de_0 \neq 0 \tag{24}$$

$$F^0(d) = \left\| \overrightarrow{DS_j^0} \right\| = A_0 \left(\frac{(\sqrt{P_t} - 2k_0\sqrt{\pi} d)}{(\sqrt{P_t} + 2k_0\sqrt{\pi} d)} \right) \tag{25}$$

The variables A_0 and de_0 set response magnitude and equilibrium distance, respectively. Also, A_0 has units of meters/second and limits the upper and lower bound on the maximum possible 'desired response' for channel-0. The channel-1 behavior transfer function uses the same equation (Equation 25). In Figure (3.15), A_0 and de_0 are set to one $\frac{meter}{\text{sec}}$and eight meters, respectively. For channel-1, A_1 and de_1 are set to 1.4 $\frac{meters}{\text{sec}}$and 0.5 meters, respectively. Since the centers of two satellites are never less than one meter apart for the constellation modeled here, the maximum repulsion for channel-0 is 0.777 meters/sec. The limit

$$\lim_{d \to \infty} A_0 \left(\frac{(\sqrt{P_t} - 2k_0\sqrt{\pi} d)}{(\sqrt{P_t} + 2k_0\sqrt{\pi} d)} \right) = -A_0 \quad \text{for } \{ \, de \in \mathcal{R} \, \} \tag{26}$$

and similarly, the limit

$$\lim_{d \to -\infty} A_0 \left(\frac{(\sqrt{P_t} - 2k_0\sqrt{\pi} d)}{(\sqrt{P_t} + 2k_0\sqrt{\pi} d)} \right) = +A_0 \quad \text{for } \{ \, de \in \mathcal{R} \, \} \tag{27}$$

guarantee that the demands placed on the control system after DS^0 and DS^1 are combined into the 'compromise' equation (Equation 20) never exceed $(A_0 + A_1)$ regardless of the incident power or lack thereof. Negative values for F0 imply the opposite of repulsion, *attraction.* Repulsion and attraction complement each other during the swarming phase. If an agent strays too far from the constellation, then received power figures drop, the compromise equation turns negative, and the agent is attracted toward the most intense source of light.

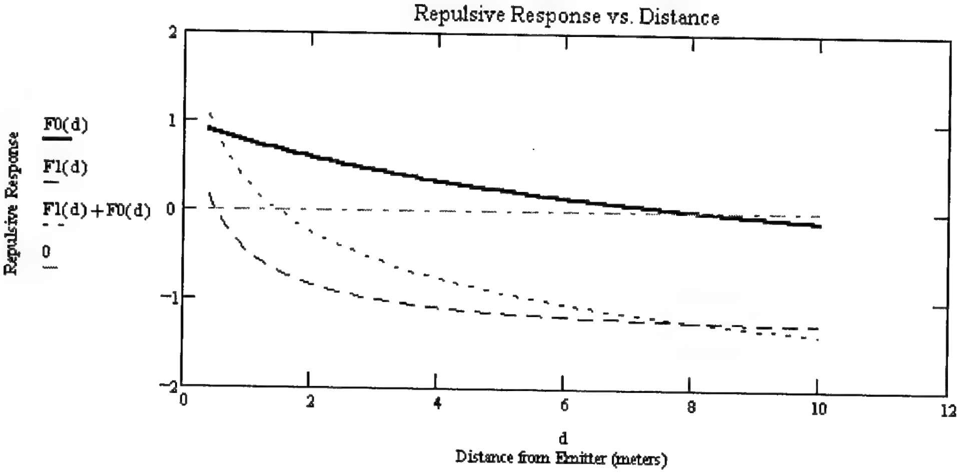


Figure 3.15 Repulsive response (m/s) vs distance (meters) from a single emitter: channels CH-0, CH-1, and (CH-0 + CH-1)

If the agent is extremely distant from the constellation and light levels are near zero, then the compromise equation approaches -($A_0 + A_1$); maximal attraction. The limiting cases of the compromise equation drive the impulse thrusters to their peak operating points. Results demonstrate limiting cases of the compromise equation at the state transitions Γ_0, Γ_1, and Γ_2 only. Thrust magnitudes stabilize at 10 to 20% of the maximum allowable between state transitions. Although the repulsive response vs. distance for the 2-satellite case yields a lower bound on the values for de_0 and de_1, this relation need not apply for the N satellite case and there is no guarantee that face activation will overpower social interaction for any structural configuration. (Later, the flat-field method is used to determine approximate figures for A and de). Equation (19) reduces to

$$F_c(P_r) \quad = \quad A_c\left(\frac{P_r^c - k_c}{P_r^c + k_c}\right) \quad \text{c} \in \{0,1\} \tag{28}$$

in scalar form. Given the same values for A and de as above, we plot the *repulsive response vs received power* in Figure (3.16)

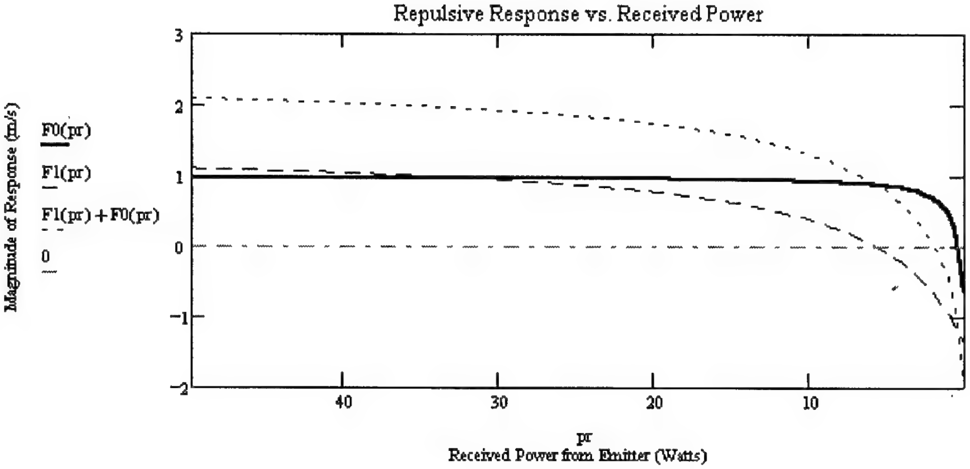


Figure 3.16 Repulsive response (meters/sec) vs received power (mWatts) for channels: CH0, CH1, (CH0 + CH1)

The independent axis now expresses power in Watts with power decreasing from left to right (as is the case with increased distance). Notice that equilibrium occurs in positions of low power reception. If the equilibrium distances are set beyond the point where environment noise obscures the lowest power reception possible, then it is impossible to achieve an equilibrium distance. Alternatively, if equilibrium distances are set too low, then power levels are high and the summation of F0 and F1 lies in a linear region. A linear region makes it exceedingly difficult for one satellite to dominate in achieving for a position. The optimal choice of values for A and de are those that place F0 and F1 in a region of non-linearity between the two aforementioned bounds. The optimal region in Figure (3.16) is between $\mathrm{Pr}_{\mathrm{max}}$ = 10.0 Watts and $\mathrm{Pr}_{\mathrm{min}}$ = 0.1 Watts, assuming that the noise σ^2 is much less than $\mathrm{Pr}_{\mathrm{min}}$. However, selecting optimal values for A and de is not an exact science and functional parameters are ultimately selected by trial and error, implying that genetic algorithms may be used successfully to optimize the system based on the given environmental constraints.

3.4.3 The Flat Field Solution

The Flat Field solution is used to estimate the behavior function parameters A and de for a constellation of N satellites. The 2-satellite approximation does not hold for structures with more than 2 satellites, because the received power now includes multiple radiators and de no longer represents a distance. Thus, the concept of distance is discarded and the goal is now to find values for A and de that guarantee structure formation. To this end any single structural element surrounded by a field of repulsive structural elements must be capable of drawing in any single constellation satellite. If this scenario is not possible, then in certain instances the desired structure fails to reach completion. While no closed form solution for A and de is available, the minimal difference between DS^0 and DS^1 is apparent.

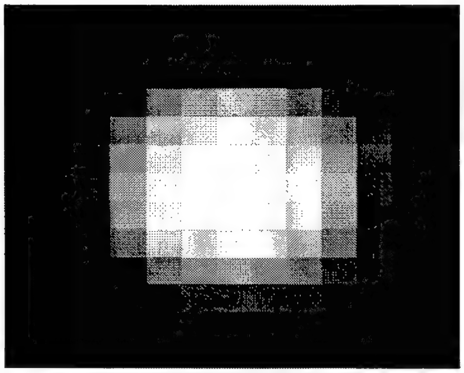


Figure 3.17 Received light intensities from a flat field of structural satellites.

Figure (3.17) illustrates the individual transmission contributions from a flat field of radiators. The net received power incident on a constellation satellite is determined by summing the elements of this matrix. The particular flat field consists of 169 satellites, each radiating 100 mWatts of power; transmitters at greater off nadir angles appear less bright. Figure (3.18) illustrates the increase in power on a receiver of varying distances from the flat field. The probability that any one satellite in a constellation will happen upon a flat field of 3,025 inactive satellites, with only one

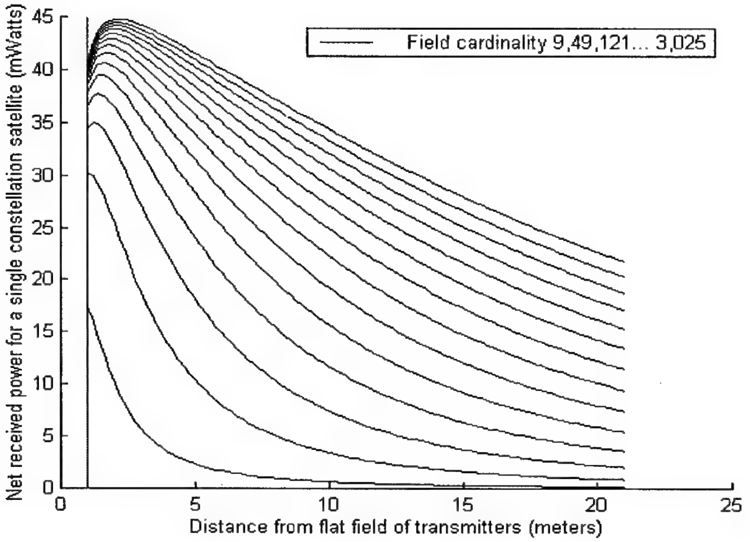


Figure 3.18 Recieved power for a single constellation satellite vs the receiver's distance from the flat-field: *Ensemble over flat-fields of cardinality {9,49,121...3025}*

active satellite in the center is very low. It follows from Figure (3.18) that the peak repulsive force for such a flat-field is a function of 45.0 mWatts ($P^1_{r\,\max}$) at a distance of 2.5 meters ($d_{\max}$). Therefore, A_0, de_0, A_1 and de_1 are selected to satisfy

$$A_1\left(\frac{P^1_{r\,\max}-k_1(de_1)}{P^1_{r\,\max}+k_1(de_1)}\right) \quad > \quad A_0\left(\frac{P^0_r(d_{\max})-k_0(de_0)}{P^0_r(d_{\max})+k_0(de_0)}\right) \tag{29}$$

If we wish to bound the limiting cases of the *compromise* equation (Equation 20) to a reasonable velocity, then a second constraint

$$A_1\left(\frac{P^1_r-k_1(de_1)}{P^1_r+k_1(de_1)}\right)+A_0\left(\frac{P^0_r-k_0(de_0)}{P^0_r+k_0(de_0)}\right) \quad < \quad \overrightarrow{DS}_{\max} \quad \forall\,\{P_r>0\} \tag{30}$$

is added. The margin by which Equation (29) is satisfied remains a matter of preference: $\overrightarrow{DS}_{\max}$ is constrained by the maximum operating point of the satellite thrusters. If a satellite is given the ability to overcome large social repulsive forces when presented with small structural attraction forces, then the combination can overwhelm local social repulsive forces. This situation is evident from results obtained in Chapter 5. Satellites become densely packed during the initial structure formation phase due to the large number of active faces transmitting on channel-1.

3.4.4 The Four-Post Face Activation Algorithm

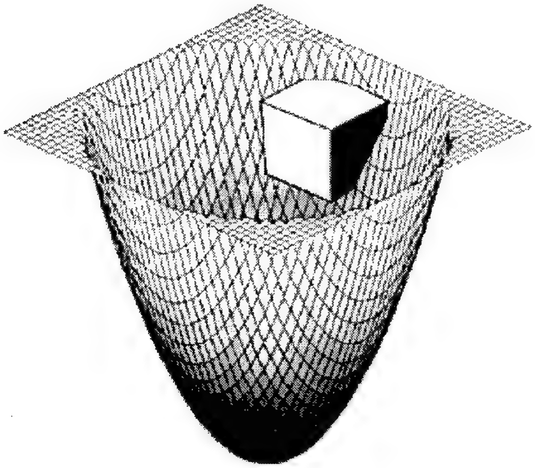


Figure 3.19 The intersection of a satellite and a surface is used to switch facial transmitter channels. Here, an active face [dark] is switched on to attract swarming agents.

Solving the inverse problem described in Chapter 1 requires genetic algorithms. Since genetic solutions require a separate rule space evolution every time the user changes structural designs, a faster method is required. The *four-post* method converts global blueprints into local rules without sacrificing robustness. Certain *apriori* knowledge is used. First, satellites in the constellation are capable of communicating digital information. Second, the structure is arranged in a regular lattice and satellite widths are known precisely. Furthermore, precise relative position information (x_0, y_0) can be transmitted from one satellite to the next within the resultant lattice structure. Subsequent satellites attach and receive position updates from local neighbors.

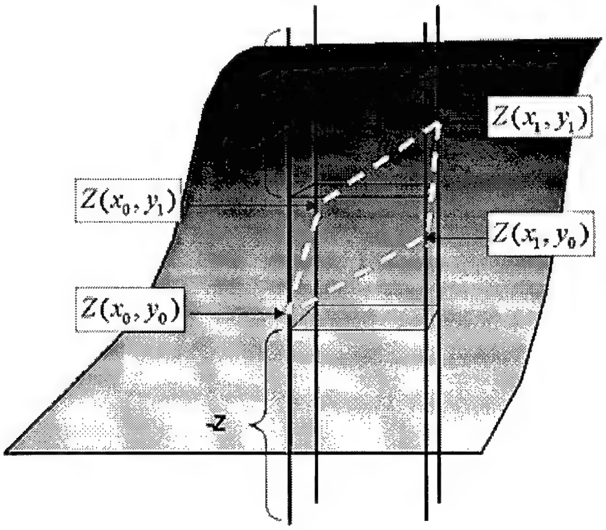


Figure 3.20 Four-post function evaluation.

Although the satellites are symmetric, 'up' can be differentiated from 'down' by assuming a Z axis. Agent 0 starts the convention, and every future satellite adopts it. After a satellite attaches to the structure, a set of global blueprints are uploaded in the form of a single valued function. Facial channel switching algorithms are developed for parametric equations, paths, and gradient vector fields. However, a wealth of single valued functions are available to choose from and investigate.

Figure (3.19) illustrates the intersection of a cube with an imaginary single valued function (structural blueprint). Agents use their position relative the structure to activate faces (transmit from CH-0 instead of CH-1). The X-Y coordinates of four posts corresponding to the four corners of a satellite are determined. Given the current relative structural satellite position (x_0, y_0, z_0), these four positions

$$\left|\begin{array}{ll} p_a = (x_0 - \frac{w}{2}, y_0 - \frac{w}{2}) & p_b = (x_0 - \frac{w}{2}, y_0 + \frac{w}{2}) \\ p_c = (x_0 + \frac{w}{2}, y_0 + \frac{w}{2}) & p_d = (x_0 + \frac{w}{2}, y_0 - \frac{w}{2}) \end{array}\right|$$

are fed into the current structural blueprint. One possible surface is,

$$Z(x,y) \quad = \quad 5\sin(x) + 6\sin(x)\cos(y) + 7\cos(y) - 10.80$$

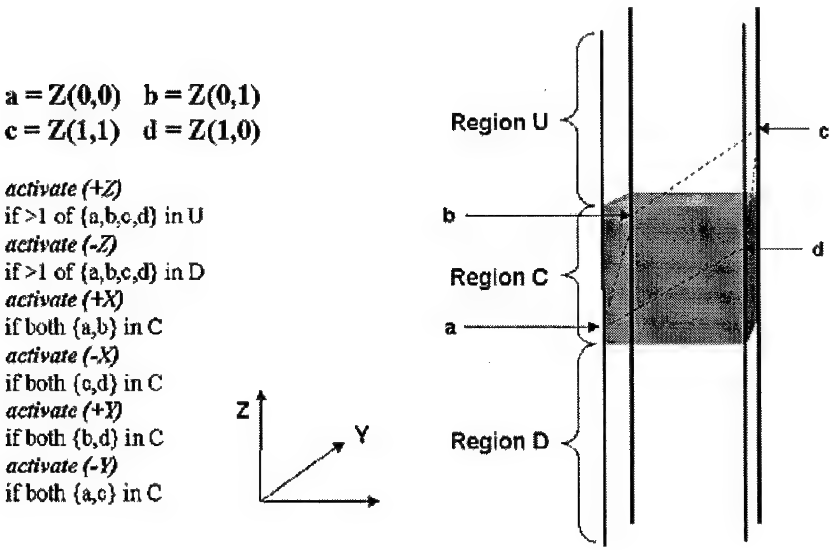


Figure 3.21 *Four-post* global-to-local face activation algorithm.

and when evaluated at four corner points yields four 'altitude' scalars

$$a = Z_a(x,y) \quad b = Z_b(x,y) \quad c = Z_c(x,y) \quad d = Z_d(x,y)$$

that describe a patch in three dimensional space. Faces are activated based on the algorithm:

$$\begin{aligned}
\pm Z &: \ any\ two\ of\ \{a,b,c,d\} < \left(z_0 + \frac{w}{2}\right) \\
+X &: \ if\ any\ one\ of\ \{a,b\} : \left(z_0 - \frac{w}{2}\right) < \{a,b\} < \left(z_0 + \frac{w}{2}\right) \\
-X &: \ if\ any\ one\ of\ \{c,d\} : \left(z_0 - \frac{w}{2}\right) < \{c,d\} < \left(z_0 + \frac{w}{2}\right) \\
+Y &: \ if\ any\ one\ of\ \{b,c\} : \left(z_0 - \frac{w}{2}\right) < \{b,c\} < \left(z_0 + \frac{w}{2}\right) \\
+Y &: \ if\ any\ one\ of\ \{a,c\} : \left(z_0 - \frac{w}{2}\right) < \{a,c\} < \left(z_0 + \frac{w}{2}\right) \qquad (31)
\end{aligned}$$

Furthermore, faces are not activated if they contact existing structural members. Given the correct structural blueprints, it is possible to attach and activate no exposed faces; however, this is not recommended. The maximum number of activated faces for a structural satellite is *five.* For example, if both a and c are in region C and both b and d are in regions U or C, then five faces are activated.

3.5 Structure Formation

3.5.1 Payload Generation with XPAYLOAD.M

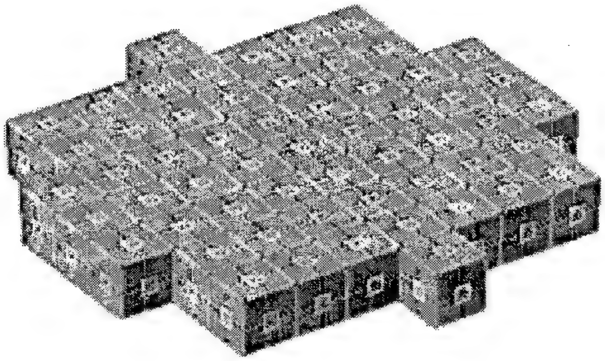


Figure 3.22 A (4,1):57 satellite cylindrical payload / *initial configuration* approximation.

Throughout Chapter 5, *Results and Analysis*, the payload configuration is generated using a NIGHTHAWK function; XPAYLOAD.M, which attempts to pack as many cubes into a cylinder of height H and radius R as possible to maximize empty payload space. XPAYLOAD.M generates a rough cylindrical approximation and specifies satellite positions in the payload. The naming convention for a cylindrical approximation payload is:

$$(\ < \ radius >, < height >) : number\_of\_satellites \qquad (32)$$

Figure (3.22) illustrates a cylinder of height 1. Once a position matrix P is generated, it is stored in the p field of the agent.phys.p initial conditions data structure.

3.5.2 State Transitions

Thrusters are fired during the Δ_{01} (Γ_0 to Γ_1) phase transition and the race to equilibrium begins. On-board software initializes Δ_{01} at $t = 0.50\,\text{sec}$: *Frame=1* ushering the fleet into the Γ_1 phase. As no global knowledge of any kind is passed between agents,

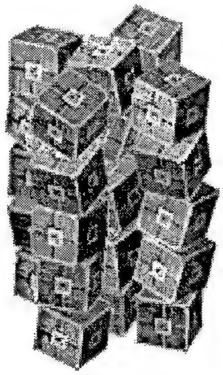


Figure 3.23 Frame 2 at 1.0 seconds: A (5,1):20 satellite configuration just after activation.

the fleet as a whole is unable to determine when maximum social equilibrium is reached and initiate the change in mood from social behavior to constructive behavior. During the period of time between Γ_0 and Γ_1, agent 0 (δ_0) in a fleet of N^0 satellites receives a data uplink. Equation (31) and a state transition time-table is passed via uplink to δ_0. The state-transition timetable is illustrated in Table (3.3). Agent zero takes action according to this timetable and seeds the state transitions. Robust leaderless behavior is sacrificed initially when δ_0 is tasked to seed structure formation. It is necessary that one agent must be first in a system of many agents and that an element of human control be allowed to direct reconfiguration Thus, leadership is required to initiate certain sequences of action. Structure formation is delayed until some δ_n is promoted if agent zero fails and improperly executes Table (3.3).

Agent zero (δ_0) only takes action at Γ_0 and Γ_1 both to break the initial payload structural configuration and free up space to move about and to initialize structure formation by flying to a location central to the swarm and evaluating Equation (31) for the first time. Excitement begins during Δ_{12} as agents swarm about δ_0 and attach to 1 of 6 active faces. The process continues much like crystal formation. One agent attaches to the structure, receives Equation (31) locally, then activates faces based on this equation.

Uplink Timetable: i = 0		
State	Release Time	Description
Γ_0^i	$t = 0.5\,\mathrm{sec}$	Payload Configuration
Γ_1^i	$t = 60.0\,\mathrm{sec}$	Social Interaction: *First Equilibrium State Reached*
Γ_2^i	$t \geqslant 60.0\,\mathrm{sec}$	Social/Constructive Behavior: *Equilibrium Transition*
Γ_3^i	*unknown*	Structural Completion: *Final Equilibrium State Reached*

Table 3.3 Earth to Satellite Uplink to Agent Zero

The assimilation frenzy continues for some time during Δ_{23}, whereas Γ_2 begins some time after social equilibrium is broken by the first face activation of δ_0. The point of demarcation for Γ_2 is when the cardinality of structure $S_i = 2$, where i signifies the *ith* structure formed using the $\Gamma_{0\to3}$ process. Here a change in i from $i \to (i+1)$ is termed a *structural reconfiguration,* which follows precisely the same $\Gamma_{0\to3}$ state transitions, only the initial structure architecture is different.

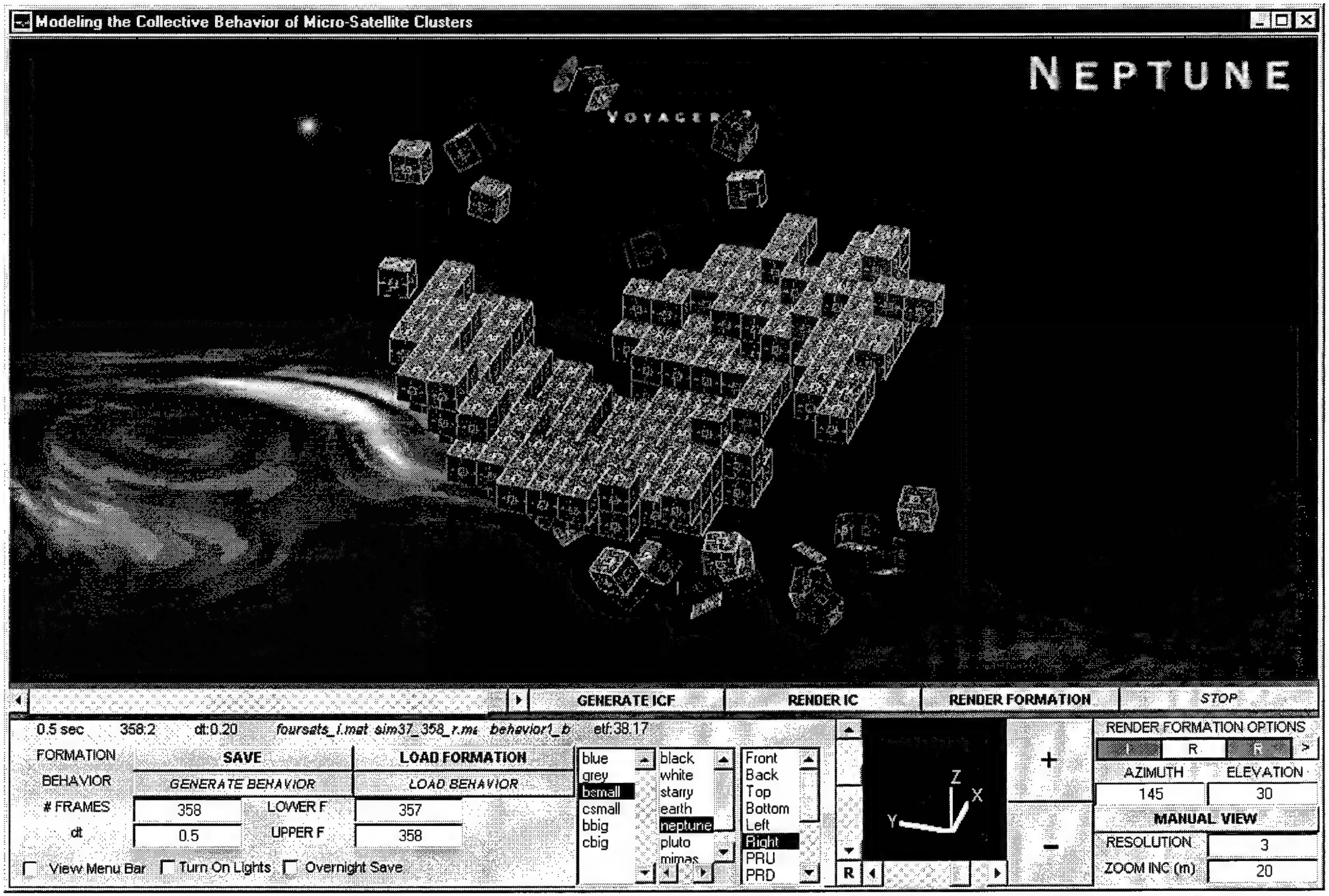
Endless phase transitions are possible provided sufficient energy is supplied. Just as carbon dioxide is capable subliming from a solid to a gas given enough energy, so too are nano-satellites with the binary and four-post algorithms. If the satellites continue to receive phase transition commands and continue to accumulate power, then the Γ_0 to Γ_2 to Γ_0 transition cycle may continue perpetually. One question is 'Are we closer to realizingVon Neumann's Universal assembler?' The answer is 'yes,' but from an algorithms approach only. The hardware must be ultimately be capable of replication and such technology is in its infancy.

4. Model

4.1 The Structural Emergence Simulator

The Structural Emergence Simulator (STEMS) MATLAB® 5.0 Graphical User Interface (GUI) is best described as a discrete time 3D nano-satellite simulation engine. It is capable of modeling and simulating N satellites in a zero gravity environment. STEMS offers an experimental platform devoid of forces, except those applied at the agent level, and it is not influenced by external gravitation. The experimental environment is centered in a collective inertial reference frame (CRF) (Chapter 2) through which motive agents may translate and rotate. As a computational model, it is discrete in nature: every *dt* seconds a new frame is generated based on the last. The decisions made by multiple satellites between frames are relatively simple when compared to the complex lattice structures they create. Although the KMS standard is used in STEMS, this thesis explores a *scale-invariant* problem. The standard dimensions for a satellite are considered (1 x 1 x 1) meters, but only to define a relative metric with which to delineate velocities. Satellites may be nanometers or kilometers on a side, i.e., dimension is only a function of existing technology, mass, and available power.

STEMS consists of two major control panels, the *Main Application User Interface* (MAUI) and the *Generate Initial Conditions Interface* (GICI). MAUI parents all other control panels and User Interface (UI) controls. It consists of multiple *list-boxes, sliders, edit-boxes, check-boxes* and *push-buttons.* The interface is MATLAB® 5.0 compliant (a 5.0 to 5.2 upgrade is due for release by the year 2000). The MAUI control panel is capable of saving and loading simulations from file. Unfortunately, simulations can consume 10-50 Megabytes of data for constellations of 100-400 satellites over 70 to 300 frames. Simulation data is stored in the environment as a data structure termed RUNN (RUN is an existing function).



Modeling the Collective Behavior of Micro-Satellite Clusters

0.5 sec 358:2 dt:0.20 *foursats_i.mat sim37_358_r.mat behavior1_b* etf:38.17

FORMATION

BEHAVIOR

FRAMES 358

dt 0.5

LOWER F 357

UPPER F 358

☐ View Menu Bar ☐ Turn On Lights ☐ Overnight Save

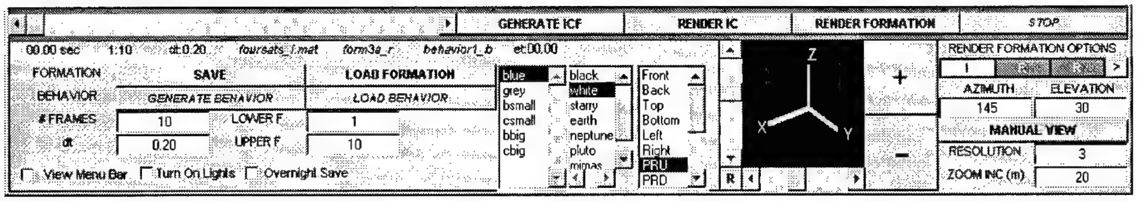
AZIMUTH 145

ELEVATION 30

RESOLUTION 3

ZOOM INC (m) 20

Figure 4.1 *The STEMS Main Application User Interface (MAUI)*



00.00 sec 1:10 dt:0.20 *foursats_i.mat* *form3a_r* *behavior1_b* et:00.00

# FRAMES 10 LOWER F 1

dt 0.20 UPPER F 10

□ View Menu Bar □ Turn On Lights □ Overnight Save

RENDER FORMATION OPTIONS

AZIMUTH	ELEVATION
145	30

RESOLUTION 3

ZOOM INC (m) 20

Figure 4.2 The Main Application User Interface (MAUI) control panel

The RUNN data structure contains sub-matrices with satellite ephemerides and run-time information, such as the frame interval (dt) and the number of simulation frames (M).

The Generate Initial Conditions Interface (GICI) (Figure 4.5) child interface is called when the **GENERATE ICF** button, positioned at the upper center of the main control panel, is depressed. GICI generates and edits the initial conditions required to run a simulation. Payload and all initial ephemeris information is entered and/or edited in GICI. GICI it is also capable of initializing the environment display variables. After the initial conditions are loaded, GICI can be closed and a simulation started in MAUI by depressing **RENDER IC**. Both GICI and MAUI are designed with the philosophy that "data interaction and realistic visualization increase both the *validity* and *quality* of experimental results."

Filenames are displayed below the horizontal frame slider and above the **SAVE /LOAD FORMATION** push-buttons. Current initial conditions are displayed as '_i.mat' files and stored simulations are displayed as '_r.mat' files. Default behavior is stored in the file 'behave1_b.m' and modified directly using the MATLAB® M-file editor or any text editor. Display parameters are selected from three list-boxes positioned at the center of Figure (4.2). Satellite rendering, background rendering, and camera position options are provided for realistic data visualization. The right-most third of the main control interface contains zoom and scene rotation controls. Mouse and manual scene rotation is possible, as is mouse controlled positive or negative zoom control. Finally, the red/green state controller allows the user to switch rendering modes from *Initial data generation* to *Replay from file* (I to R).

4.1.1 List-box Functions

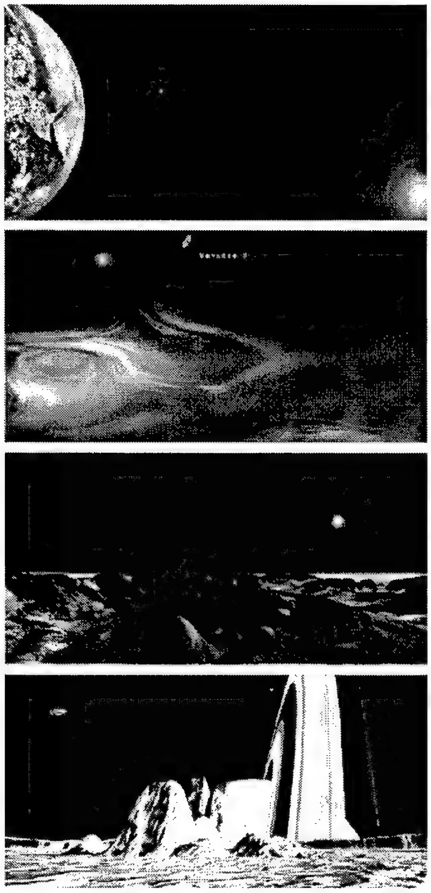


Figure 4.3 Main STEMS GUI background set: *Earth [top], Neptune [upper middle], Pluto/Charon [lower middle], Mimas [bottom] [* images 2,3, and 4 may be found on the JPL webpage*]*

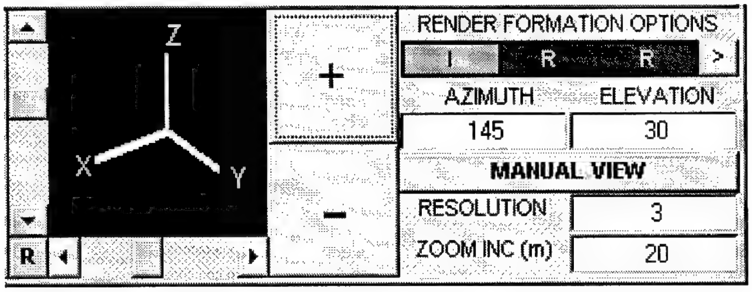
Three list-boxes are used in the MAUI control panel (Figure 4.2). The first list-box changes the conceptual satellite rendering mode. The first satellite color option is *blue.* When option *blue* is selected, it sets the current display variable (D BIT) in resident memory to zero so that the next run renders each satellite in blue. The second list-box option is *grey* rendering. It is functionally equivalent to blue, except grey prints with greater contrast on black and white (b/w) printers. Color images are mapped to darker greys and contrast information is lost when printed in b/w, in which case the option *grey* with a white background yields best results.

The third list-box item is termed *bsmall*, which refers to a black and white conceptual bitmap image.in the **c:\MATLAB\thesis\panels\** directory. If this file structure (see section *File Structure*) does not exist or if filenames are changed, then MATLAB® returns an error message. *Bsmall* is 50 x 50 pixels in dimension and as in option two, prints with higher contrast than the color version on b/w printers. The MATLAB texture mapping routine is exceedingly slow (on an Intel® Celeron 450, 128 MB RAM) in rendering scenes of 100 or more satellites. List-box item number four is a color version of list-box item number three. List-box items *bbig* and *cbig* are higher resolution (128 x 128) versions of list-box items *bsmall* and *csmall.* These options should be avoided unless four or fewer satellites are in the model. For *bbig* and *cbig,* MATLAB must calculate the position and shading for 393,216 x N^0 (128 x 128 x 6 x 4 x N^0) separate corner vectors that define four-cornered face patches.

If sufficient RAM and computing power is available, rendering with the high-resolution option produces excellent imagery. The NRU camera position, coupled with the grey satellite rendering option on a white background, is best for data visualization. MATLAB quickly renders and shades with this configuration. However, a decrease in speed is realized when a complex background (Earth, Neptune, Charon, or Mimas) is selected instead of the white or black options.

The *center list-box* contains seven conceptual backgrounds (Figure 4.3). Black and white backgrounds offer the best data/scene contrast of any single satellite option and are best used for constellation data analysis. list-box item (starry) *four* is identical to list-box item one (black). The conceptual earth background (Figure 4.3), [top]) illustrates a realistic juxtaposition for the satellite constellation. However, this realism should not imply that the effects of orbital mechanics are modeled in STEMS. The remaining three backgrounds are artists conceptions [www.jpl.com].

The *right-most* list-box provides quick transitions to 15 unique camera perspectives. list-box option *front* looks down the -X axis toward the YZ plane. Similarly, *back* looks down the +X axis at the YZ plane. *Top* looks down the -Z axis toward the XY plane and *bottom* looks up the +Z axis at the XY plane.



AZIMUTH 145

ELEVATION 30

RESOLUTION 3

ZOOM INC (m) 20

Figure 4.4 The Camera Perspective User Interface Control Panel (CPCP)

Option *left* points the camera down the -Y axis and *right* looks down the +Y axis at the YZ plane. The remaining eight camera (viewer) perspectives are coded in text strings using the set of six letters {P,N,L,R,U,D}. Each selected view points toward the origin from one of the eight {PLU, PRU, PLD,PRD,NLU,NRU,NLD,NRD} possible isometric views. Letter P stands for ‘positive’ and implies a camera move along the positive X axis. Similarly, N moves the camera down the -X axis. View letter R stands for 'right' and moves the camera along the +X axis and L moves the camera down the -X axis. Finally, U (up) and D (down) translate the camera along the ±Z axes, respectively.

Scene lighting from two scene spotlights makes NRU the view of choice. Structural satellites are clearly defined by the lighting when viewed from this venue.

4.1.2 Display Routines

At the far right of the MAUI control panel is the Camera Perspective Control Panel (CPCP), which provides manual or dynamic camera perspective adjustments in the experimental environment. The vertical elevation slider (CPCP: left) adjusts camera elevation. Clicking on the up or down arrows increases or decreases the elevation by increments set in the RESOLUTION edit-box. Azimuthal adjustments relative to the -Y axis are made in the same manner. Ten percent (of full range) jumps are possible if one clicks the left mouse button in the intermediate light gray area. For elevation and azimuth ten-percent equates to 18° and 36°, respectively.

The (CPCP) coordinate axis adjusts immediately to changing input. The central plus [+] and minus [-] buttons are zoom controls and are primarily used to fit the displayed satellite constellation within the MAUI (1000 x 500 pixel) main viewable area. The zoom increment defaults to 20 meters, but can be altered during operation by modifying the ZOOM INC (m) editbox variable. A zoom increment of 100 meters is recommended for constellations of ≈400 satellites. It is possible to adjust the manual view edit-box parameters for precise camera positioning. Finally, the red/green LED three state controller switches display modes from ‘Initial Computation’ (I) to ‘Render’ (R) on demand. One may change the **RENDER FORMATION OPTIONS** state by clicking on the [▷] push-button. To improve clarity, CPCP alters the letters I and R from white-on-red to black-on-green.

4.1.3 Model Lighting

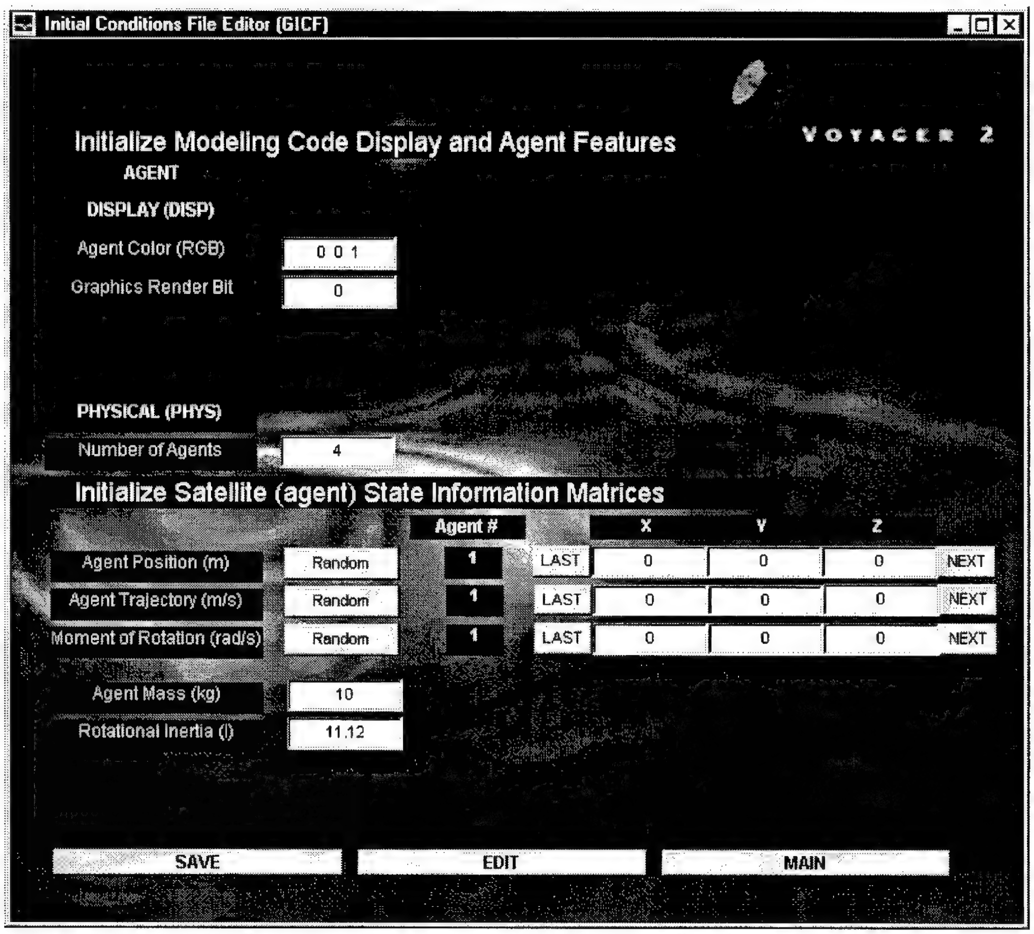
The function HIT LIGHTS.M calls several high-level MATLAB® graphics functions. First, it sets the lighting type to flat, which is optimal from a speed standpoint, because curved surfaces are never rendered in STEMS. Two spotlights with different intensities are cast on the scene from the aforementioned PRU and NLD positions. The brighter spotlight points from the PRU perspective at an infinite distance from the origin. An distant light source ensures that lighting is always valid, regardless of the structure dimension. HIT LIGHTS.M reads as

```
lighting flat
light(‘Position’,[+300 +300 +300],‘Style’,‘infinite’,‘Color’,[1.0 1.0 1.0])
light(‘Position’,[-300 -300 -300],‘Style’,‘infinite’,‘Color’,[0.5 0.50.5])
material dull
```

Finally, the material type is set to dull. Type ‘Dull’ reflects with diffuse properties and works best with the given texture maps and flat shading colors.

4.1.4 Excluded Functions

Three STEMS User Interface (UI) controls are not used. The first two are the Generate Behavior (GB) and Load Behavior (LB) push-buttons. Modifying the behavior functions is a dynamic process



Initial Conditions File Editor (GICF)

Initialize Modeling Code Display and Agent Features

AGENT

DISPLAY (DISP)

Agent Color (RGB): 0 0 1

Graphics Render Bit: 0

PHYSICAL (PHYS)

Number of Agents: 4

Initialize Satellite (agent) State Information Matrices

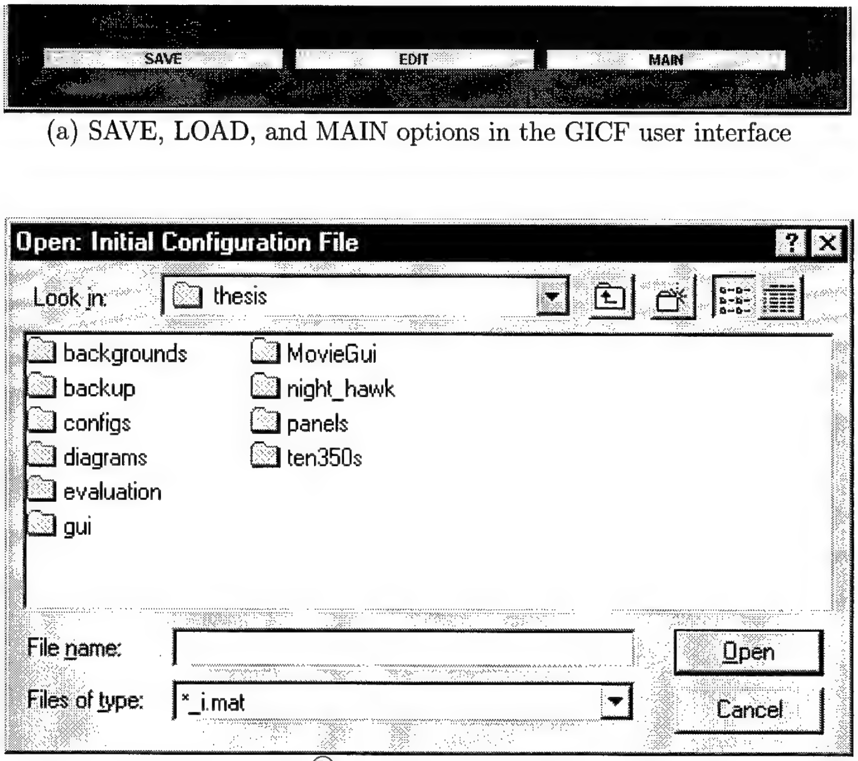
		Agent #		X	Y	Z	
Agent Position (m)	Random	1	LAST	0	0	0	NEXT
Agent Trajectory (m/s)	Random	1	LAST	0	0	0	NEXT
Moment of Rotation (rad/s)	Random	1	LAST	0	0	0	NEXT

Agent Mass (kg): 10

Rotational Inertia (I): 11.12

Figure 4.5 Generate Initial Conditions Interface (GICI) control panel: *used to generate, save, and edit initial conditions files.*

that is better left at the code level than forced into a rigid user interface. One strength of the STEMS simulator lies in the flexibility that it allows for future innovative research. Thus, the GB and LB user interface buttons (Figure 4.1) are grayed-out. The third non-functional UI control is in the RENDER FORMATION options block. The two right-most options are identical (R and R). The third option was originally an MPEG movie storage and playback option; however, this option limited the movie storage format to mpeg only. The shareware program 'SnagIt-32' (by the TechSmith Corporation) is capable of capturing movies in *.AVI format with various compression ratios and proved to be the better option.



Look in: thesis

File name:

Files of type: *_i.mat

(a) SAVE, LOAD, and MAIN options in the GICF user interface

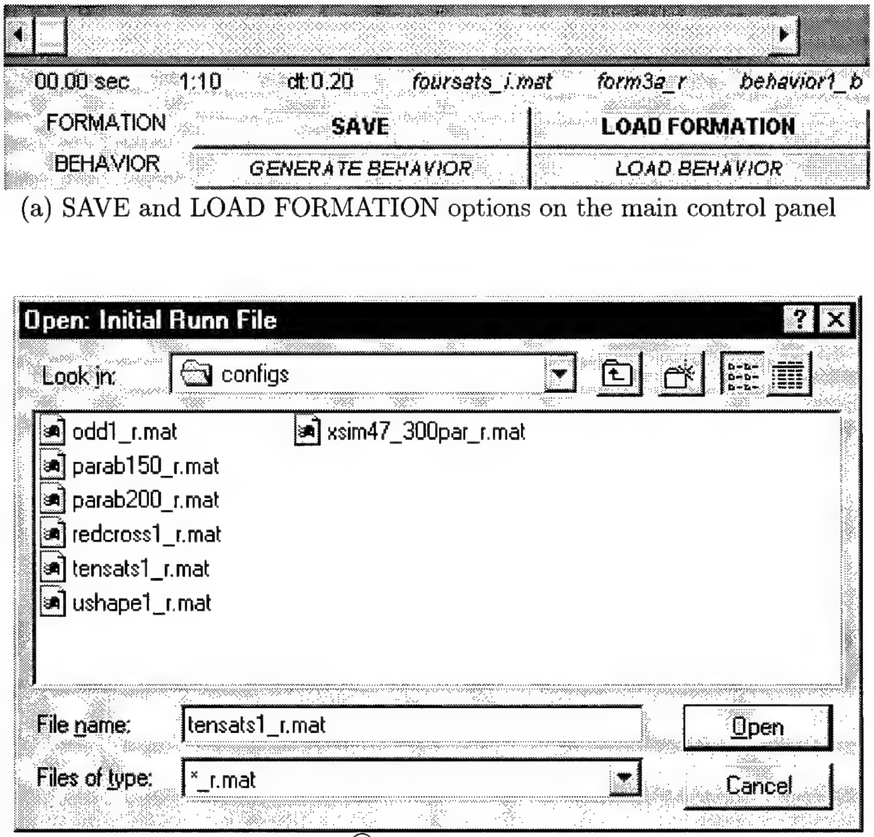
(b) MATLAB® UI file manager: GICF editor.

Figure 4.6 Generate Initial Conditions Interface file manager

4.1.5 The Generate Initial Conditions Interface (GICI)

4.1.5.1 3D Ephemerides. Modifying the initial ephemerides for an N^0 satellite constellation is made easier with GICI. Sometimes referred to as the Generate Initial Conditions File (GICF) manager, GICI allows for rapid data entry. Figure (4.5) illustrates the layout of GICI UI controls. The upper portion of the interface is reserved for display parameters. Default values are loaded automatically when GICI is opened, but one may edit the values freely. Originally, one could edit the flat shading matrix color as well. However, this function was moved from a static initial condition to a dynamic feature on the main control panel. The leftmost list-box now provides real-time selective satellite rendering.

The lower three-fifths of the GICI panel are used solely for ephemeris data entry. A 3 x 3 matrix of editboxes displays single rows of *initial position, rotation,* and *translation vectors.* The last and next push-buttons on the left and right of



Open: Initial Runn File

Look in: configs

odd1_r.mat
parab150_r.mat
parab200_r.mat
redcross1_r.mat
tensats1_r.mat
ushape1_r.mat
xsim47_300par_r.mat

File name: tensats1_r.mat

Files of type: *_r.mat

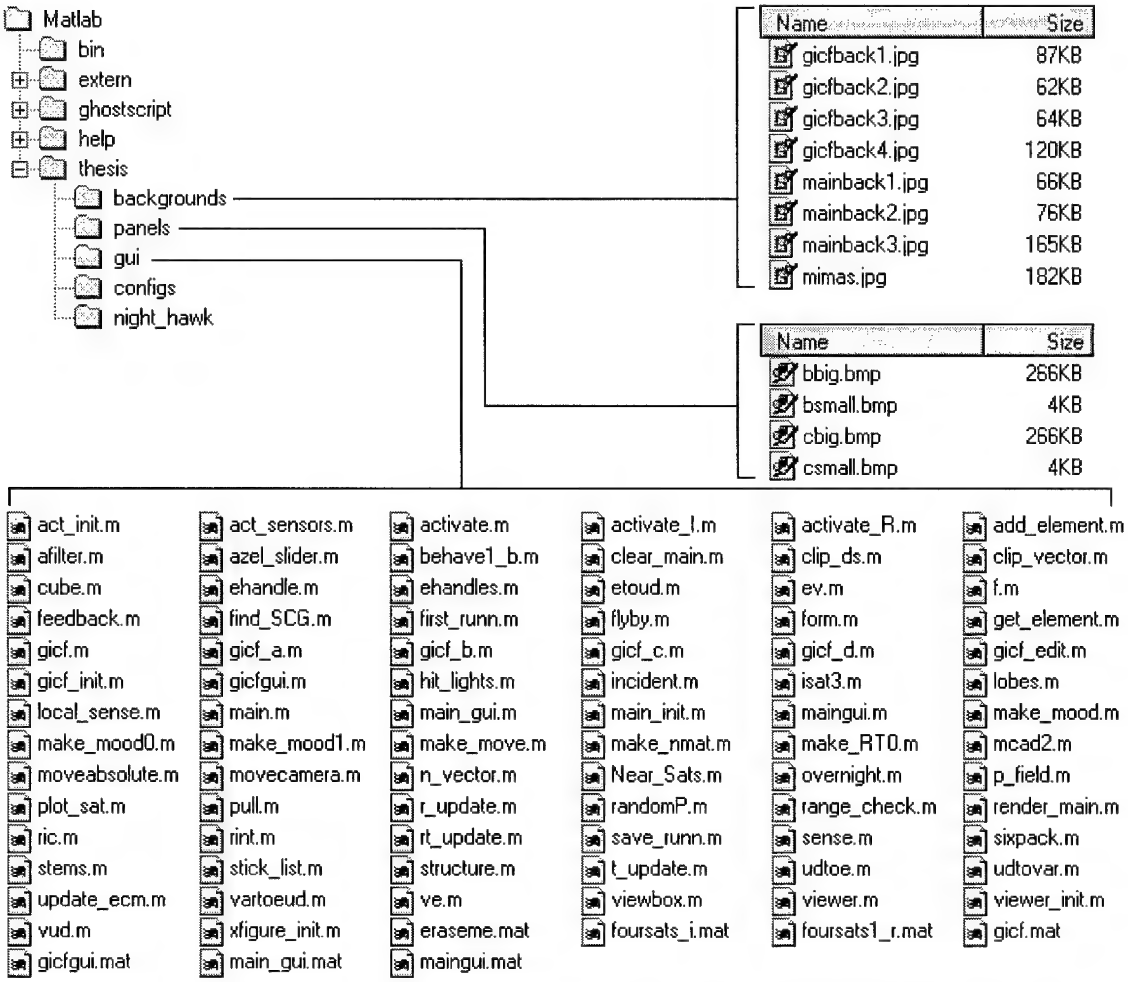
(a) SAVE and LOAD FORMATION options on the main control panel

(b) MATLAB® file manager: IRF editor.

Figure 4.7 Load and save runn file manager

each (1 x 3) edit-box array, increment and/or decrement the row counter. If **NEXT** is clicked repeatedly, thus incrementing the row index beyond the N rows specified in the number-of-satellites editbox, then the matrix is dynamically filled with zeros starting with row (N+1). If a random distribution of position, rotation, and initial velocity vectors is required, then the **RANDOM** push-buttons may be pressed to fill the corresponding matrices with a random distribution of numbers. Since the behavior function parameters developed here rely on an agent mass of 10kg, the values in the agent mass and moment of rotational inertia (I_z) edit-boxes are for record only.

4.1.5.2 Saving and Retrieving. After a set of initial conditions is entered in GICI, it may be saved. Saving stores the current initial constellation information in '_i.mat' files for later editing.



```
Matlab
├─ bin
├─ extern
├─ ghostscript
├─ help
└─ thesis
   ├─ backgrounds
   ├─ panels
   ├─ gui
   ├─ configs
   └─ night_hawk
```

Name	Size
gicfback1.jpg	87KB
gicfback2.jpg	62KB
gicfback3.jpg	64KB
gicfback4.jpg	120KB
mainback1.jpg	66KB
mainback2.jpg	76KB
mainback3.jpg	165KB
mimas.jpg	182KB

Name	Size
bbig.bmp	266KB
bsmall.bmp	4KB
cbig.bmp	266KB
csmall.bmp	4KB

```
act_init.m       act_sensors.m   activate.m      activate_l.m    activate_R.m     add_element.m
afilter.m        azel_slider.m   behave1_b.m     clear_main.m    clip_ds.m        clip_vector.m
cube.m           ehandle.m       ehandles.m      etoud.m         ev.m             f.m
feedback.m       find_SCG.m      first_runn.m    flyby.m         form.m           get_element.m
gicf.m           gicf_a.m        gicf_b.m        gicf_c.m        gicf_d.m         gicf_edit.m
gicf_init.m      gicfgui.m       hit_lights.m    incident.m      isat3.m          lobes.m
local_sense.m    main.m          main_gui.m      main_init.m     maingui.m        make_mood.m
make_mood0.m     make_mood1.m    make_move.m     make_nmat.m     make_RT0.m       mcad2.m
moveabsolute.m   movecamera.m    n_vector.m      Near_Sats.m     overnight.m      p_field.m
plot_sat.m       pull.m          r_update.m      randomP.m       range_check.m    render_main.m
ric.m            rint.m          rt_update.m     save_runn.m     sense.m          sixpack.m
stems.m          stick_list.m    structure.m     t_update.m      udtoe.m          udtovar.m
update_ecm.m     vartoeud.m      ve.m            viewbox.m       viewer.m         viewer_init.m
vud.m            xfigure_init.m  eraseme.mat     foursats_i.mat  foursats1_r.mat  gicf.mat
gicfgui.mat      main_gui.mat    maingui.mat
```

Figure 4.8 STEMS required *directory structure*, *function library*, *background image archive*, and *satellite face texture-maps.*

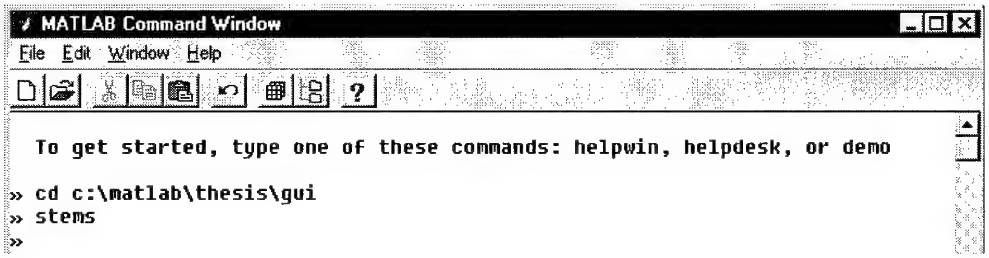
The GICF manager is called by clicking on the **SAVE** or **EDIT** push-buttons (see Figure 4.6-a) in the GICF editor. A default satellite configuration, called 'foursats1_i.mat,' is loaded into resident memory every time GICI is opened. Recall that '_i.mat' files write the AGENT data structure to resident memory. Thus, any existing current AGENT data structure is over-written with the AGENT data structure stored in 'foursats1_i.mat' when GICI is called from MAUI.

Editing loads the data structure (stored in an '_i.mat' file) into resident memory, which MAUI then recognizes as a previously entered set of initial conditions. Now, simulations may be initiated. To prevent the over-writing of memory resident initial conditions when saving, GICI does not over-write the contents of resident memory with the modified AGENT data structure. Therefore, to simulate the initial conditions entered after saving, the file is recalled by pushing the **EDIT** button and re-loading the saved file.

4.1.6 Runn File Storage and Retrieval

After a simulation is calculated using MAUI, it may be saved. Saving stores the current initial constellation information in an '_r.mat' file and allows later replaying (see LED state controller). The Initial Runn File (IRF) manager is called by clicking on the SAVE or LOAD FORMATION push-buttons (see Figure 4.7-b) on the MAUI control panel. A default simulation runn data structure is loaded into resident memory from the default 'foursats1_r.mat' when STEMS is first opened. Recall that '_r.mat' files store the RUNN data structure in resident memory. Any current RUNN data structure is over-written with the RUNN data structure when **LOAD FORMATION** is called and executed.

4.1.6.1 Required Directory and Function Library. Every function required to run the STEM simulator in GUI form is listed in Figure (4.8). A 'stems.zip' file holds the required directory structure and associated files. To run, extract the contents of stems.zip in the **c:\MATLAB** directory then open MATLAB 5.0 and type

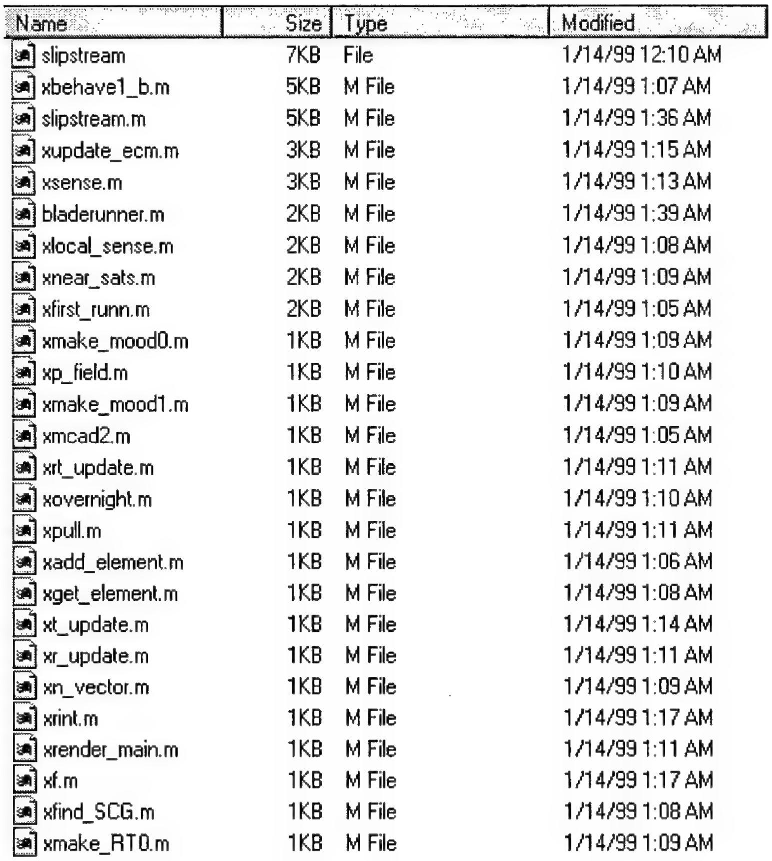


```
 To get started, type one of these commands: helpwin, helpdesk, or demo

» cd c:\matlab\thesis\gui
» stems
»
```

Figure 4.9 Running STEMS for the first time

at the command prompt (see Figure 4.9). The MAUI main interface then opens and the image of a four satellite constellation is rendered.



Name	Size	Type	Modified
slipstream	7KB	File	1/14/99 12:10 AM
xbehave1_b.m	5KB	M File	1/14/99 1:07 AM
slipstream.m	5KB	M File	1/14/99 1:36 AM
xupdate_ecm.m	3KB	M File	1/14/99 1:15 AM
xsense.m	3KB	M File	1/14/99 1:13 AM
bladerunner.m	2KB	M File	1/14/99 1:39 AM
xlocal_sense.m	2KB	M File	1/14/99 1:08 AM
xnear_sats.m	2KB	M File	1/14/99 1:09 AM
xfirst_runn.m	2KB	M File	1/14/99 1:05 AM
xmake_mood0.m	1KB	M File	1/14/99 1:09 AM
xp_field.m	1KB	M File	1/14/99 1:10 AM
xmake_mood1.m	1KB	M File	1/14/99 1:09 AM
xmcad2.m	1KB	M File	1/14/99 1:05 AM
xrt_update.m	1KB	M File	1/14/99 1:11 AM
xovernight.m	1KB	M File	1/14/99 1:10 AM
xpull.m	1KB	M File	1/14/99 1:11 AM
xadd_element.m	1KB	M File	1/14/99 1:06 AM
xget_element.m	1KB	M File	1/14/99 1:08 AM
xt_update.m	1KB	M File	1/14/99 1:14 AM
xr_update.m	1KB	M File	1/14/99 1:11 AM
xn_vector.m	1KB	M File	1/14/99 1:09 AM
xrint.m	1KB	M File	1/14/99 1:17 AM
xrender_main.m	1KB	M File	1/14/99 1:11 AM
xf.m	1KB	M File	1/14/99 1:17 AM
xfind_SCG.m	1KB	M File	1/14/99 1:08 AM
xmake_RT0.m	1KB	M File	1/14/99 1:09 AM

Figure 4.10 The NIGHTHAWK function library: *for long duration computationally intensive simulations without the overhead associated with high-resolution graphics*

The default simulation is started by clicking on the **RENDER IC** button. Ten frames are then computed and stored in the RUNN data structure. To replay the simulation, switch the render mode from I to R and click on **RENDER FORMATION**. The simulation may be saved by clicking on **SAVE**.

4.1.6.2 NightHawk Engine. The function names in the NIGHTHAWK library (Figure 4.10) are preceded by the letter '**x**' to differentiate them from STEMS library functions. The NIGHTHAWK and BLADERUNNER files are parent functions that facilitate long duration simulations specified by variables stored in NIKITA.M. The NIGHTHAWK library offers the functionality of STEMS without the computationally intensive graphics overhead.

4.1.7 Conceptual Illustrations from the Structural Emergence Simulator

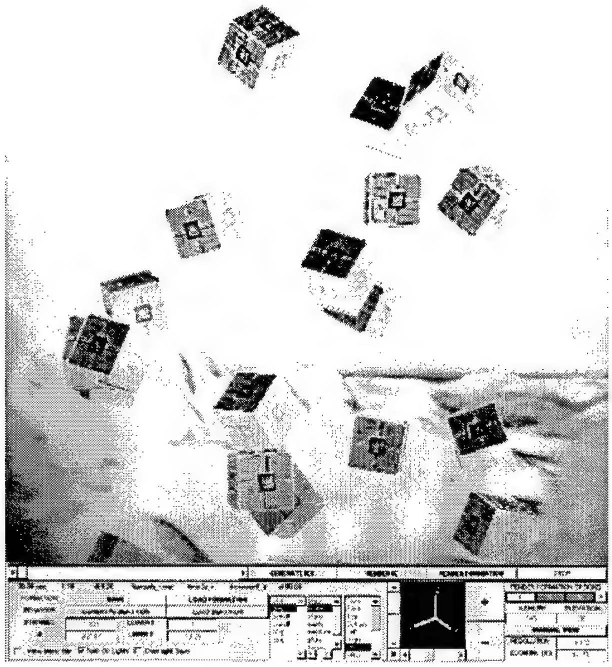


Figure 4.11 Collaborative Behavior Simulation: *"Twenty Satellites in Orbit About Charon : Autonomous Equilibrium Demonstration"* Rendered December 20, 1998. Note: this conceptual image portrays agents in an ambiguous environment. It may be on the sea-floor or in a distant galaxy. The agents may be constructing the next generation of submarines, sea-floor habitats, or a human habitat below the icy surface of Europa. Construction in harsh environments requires both autonomy and robustness.

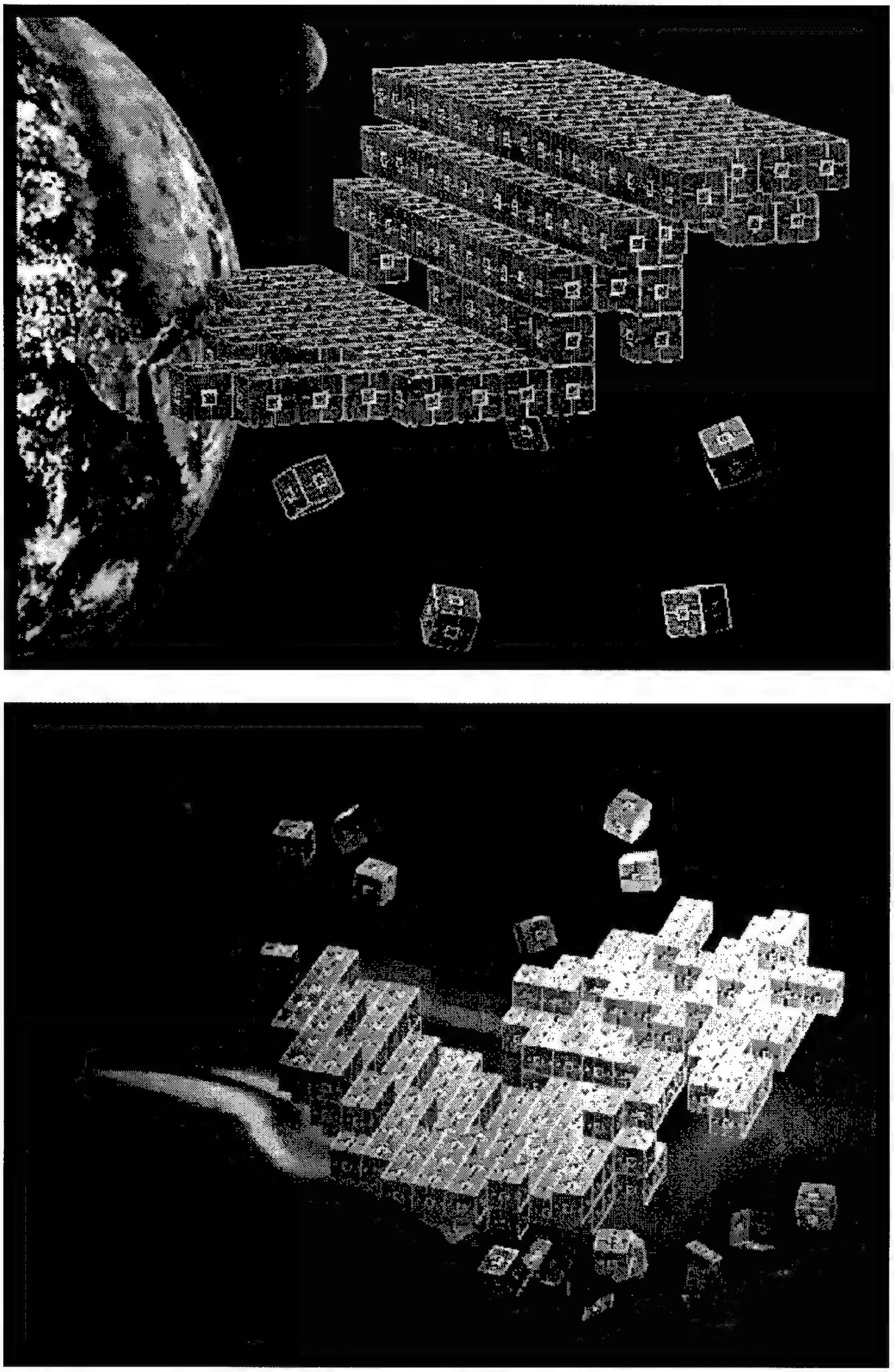


Figure 4.12 Conceptual illustration of inverted half-cylinder construction in earth orbit: *created on Dec 26, 1998* [upper plate] Conceptual illustration of paraboloid construction in a scale-less environment: *created on February 17, 1998* [lower plate]

5. Results and Analysis

5.1 Introduction

5.1.1 About the Battery of Tests

One goal of this thesis is to demonstrate that behavioral principles found in nature can be logically extended to autonomous construction [6,13,14,23,28] in space. This goal is addressed by demonstrating structure formation in zero-gravity. The results presented in this chapter extend beyond proving this goal and coding the generalized Structural Emergence Simulator (STEMS): they quantify the behavior of a chaotic system of swarming satellites. Final results dictate that, 'identical autonomous satellites can build a pre-designed structure with a set (Table 5.1) of minimal physical characteristics.'

MISSION ESSENTIAL CAPABILITIES FOR AN AGENT	
Type	*Source* and *Description*
Power	from Solar, Cold Gas, and/or Fuel Cells, etc.
Power Storage	in Batteries and/or Fuel Tanks etc.
Motive Ability	less than 10 (kg·m/s) impulse for a 10 kg satellite
Darkness	on line-of-site transmission channels 0 and 1
Intelligence	sufficient to evaluate a single-valued function
2-Channel Line of Site RX/TX	for *Magnitude* transmit and receive only
Digital Uplink	for ground induced re-configuration

Table 5.1 Minimal set of satellite characterisitics required to achieve structure formation in zero-gravity

Table (5.1) does not allow precise GPS positioning or digital data transmission between constellation satellites. Agents are allowed only a relative position update and the coefficients to a single valued (or parametric) equation once assimilated into the structure. These constraints are sufficient for a maximally robust swarm of agents at a cost. Results gathered in eight case studies illustrate one problem with an emphasis on locality and robust behavior. Given only local information, agents are unaware of the 'big picture.' Coherent structure formation [9,45,46] is exceedingly difficult to ensure. Holes and spikes of various magnitudes tend to appear in the resultant structure. In the case of a paraboloid, one side often claims more agents than the other. Three potential solutions exist. The first is an agent death scenario, the second is a solution to the optimal noise floor, and the third is structure rotation about a precessing moment. The scale problem should be addressed in future work. The costs of solving the scale problem are added intelligence and communication ability and both can sacrifice robustness.

Distributed systems are very complex when viewed as a whole. It is difficult to predict future behavior in advance or to determine how local swarming behavior functions must be modified to optimize certain conditions. However, overall trends can be analyzed and interpreted to give insight into the underlying behavior. The number of parameters available for modification is exceedingly large and it is unfortunate that we must select only a few.

A battery of eight tests is devised to interpret the following 3D time-series illustrations. The battery of tests answers a number of questions. Two parameters, out of several hundred, are selected and used as axes for a exhaustive search analysis. The first is *structure architecture* and the second is *additive position error.*

5.1.2 Structure Architecture

The style of architecture affects construction rate and structural coherency. Better coherency [45] implies construction closer to an expectation given some style of architecture. Example: consider a structure of architecture type 'paraboloid.' Agents first execute a burst maneuver and reach a point of equilibrium. Here, satellites are in a state of equipotential. Two possibilities exist for an agent in equilibrium. First, the agents internal to the swarm receive receive equal power on all six facial transducers, and second, some agents experience group cohesion as members of the constellation outer perimeter. The same structure is never built the same way twice, because agents collapse from spherical equilibrium differently each time. Spherical structures require less time to build, because agents travel less far before assimilation. Long structures are more difficult to build if the ends extend beyond the equilibrium configuration.

Power minimization is a concern, but is not clear that an initial equilibrium configuration with a large radius is required as following simulations illustrate. We know inter-satellite distance is required. As physical entities, satellites are unable to move and reposition effectively unless they are in a spatially homogenous distribution. Hence, the need for a burst to equilibrium from the densely packed payload configuration. Due to the complexity of the structure formation problem, i.e., checking physical exclusion conditions, agent attachment scheduling, and other reality parameters, symmetric and coherent structure formation is problematic. Agents attach asynchronously if they are within range of an active face. The most probable result is uneven structure formation or structure formation with holes in areas that should be filled. Incoherence is a function of both the architecture (*paraboloid, periodic, sinc, gaussian*, etc.) and the level of noise. A flat sheet forms in a different manner than other functions, because of its spatial distribution in space. These spatial differences coupled with differences in moments of rotation are factors deciding ‘successful’ structure formation or incoherent failure. One desires more of the former and less of the latter. If millions of dollars and mission success are at stake, we had better be confident that the architecture style is viable. A number of different architectures are simulated to determine the influence of architecture and noise on rate and coherency.

5.1.3 Certainty in Position

Thruster noise is the second variable of interest. Error affects agents in four primary ways. The first is via reception, the second is through physical perturbations (solar wind, exhaust gasses, etc.), the third is through calculation precision error, and the fourth is through thruster error. Others exist, but those four sources of error are notably significant. Sensor error is potentially the most devastating if it is magnified by the control system. If the fleet happens upon the light of day or a passing meteorite, then noise in the form of light (on channels 0 or 1) can mislead the satellite. Agents respond as stochastic automata (SCAs) to local neighbors in the constellation. After assimilation they change classes to deterministic automata (DCAs) [57,58] and are subject to virtually no uncertainty. Noise can have detrimental or beneficial effects upon the structural formation mission. Future results indicate that it is most probable that noise does not effect the mission, rather, it is required for efficiency.

Absolute statements regarding the probability of successful structure formation are not available. Thus, is not possible to state, 'noise of variance greater than threshold X will cause catastrophic failure' or 'paraboloid structure formation always requires less time than 2D-sinc structure formation.' However, trends do exist and are presented in the following analysis batteries. In collaborative systems noise is passed from agent to agent and throughout the swarm. The behavior of individual agents either dampens the system noise, amplifies it, or keeps it in check. Catastrophic failure is the result of under-damped noise [44,49,53]. Sub-optimal performance is achieved with critically damped system noise, leaving only one option, over-damped system noise. If satellites break cohesion and jettison into space when confronted with noise, then a normally stable system turns chaotic and breaks equilibrium. In automata theory, the transition from ordered action to chaotic action is an entire rule transition, i.e., Wolfram II to IV [31,57,58]. Cumulative internal errors ultimately pass through the control system and reach the individual satellite thrusters. Some thrusters deal with rotation and others deal with linear translation. Linear thrusters *do* couple to rotational moments and rotational thrusters couple to linear moments. Neither are ideal and both induce uncertainly in position.

The generalized Structural Emergence Simulator was written to study the trends of a distributed satellite system. In simulation, the cumulation of noise is not modeled by adding the noise due to reception error, physical perturbations, calculation error, and thruster error. Instead, the sum of these additive sources of noise is combined into one Additive White Gaussian Noise (AWGN) $\mathcal{N}_{xyz}([0,0,0],\sigma^2_{xyz})$ term and added to the net linear velocity vector. The probability distribution of uncertainty in position is assumed to be normal, because a Gaussian is the maximal entropy distribution. Imagine, in three dimensions, a vector with a cloud of uncertainty in position about the tip. When an agent fires its thrusters it does not end up in the position intended. Not only does it *not* go in the proper direction, but a coupling of linear to rotational moment is assumed and the satellite spins about a slightly different moment. Every satellite is synchronized to the same clock during the payload phase. Thus, subsequent thruster firings are near synchronous in nature. The effects of noise on structure formation coherence can be studied and conclusions can be drawn by varying the σ^2_{xyz} parameter of cumulative uncertainty in position $\mathcal{N}_{xyz}$.

5.1.4 Strong Social Equilibrium

A major problem encountered during experimentation with behavior rules, frequency selection, impulse magnitude, etc. is strong equilibrium. After Γ_1, satellites seek an equilibrium state. As previously mentioned, equilibrium state exists as a balance between interacting social forces. Prior to structure formation, two social forces are at work; social attraction and social repulsion. Social attraction [27,28,47] occurs when the net light intensity on an agent is below a threshold set by mood function 0. For example; consider 1.13 mWatts of received light power an equilibrium threshold. Reception of 1.0 mW causes an agent to feel *lonely* and it tends toward the well of low power. Reception of 1.15 milli-Watts sends the agent into an *uncomfortable* mode and it runs from the source of greater nominal intensity Action is taken based upon a nonlinear function (see Chapter 4, *Methodology*) of the received power. Strong equilibrium is the result of these social interactions.

If satellites are left alone to fly until satisfied with their respective positions, they fall into strong local (spatial) minima. In this state, it helps to imagine agents as elements suspended in the center of soap bubbles. Like soap bubbles, agents using the behavior algorithms described in Chapter 4, *Methodology* stick together at a distance. Intuition suggests that perfect equilibrium is a state with spatial symmetry, but the evidence demonstrates that this is not necessarily the case. A trend that STEMS demonstrates during equilibrium is spherical tendency. Satellites communicating over one channel tend to assemble in large spherical clusters. Satellites on the outer shell of this cluster are attracted to the group as a whole, but still repel each other. The result is analogous to surface tension. Surface satellites actually bind and compress internal satellites. In response, internal satellites seek to positions of equipotential. Satellites in these positions receive the same power on each of their six facial sensors and cease firing thrusters. The magnitude of this power is of no consequence. It follows that this point of strong equilibrium occurs when minimal noise is added to the system. Thrusters fire precisely as planned and agents reach points of perfect equipotential. In this state, no single agent has the will to move until disturbed by outside forces. Prior to the implementation of two reality-inspired solutions, forming large structures was exceedingly difficult due to strong cohesive forces.

5.1.5 Breaking Strong Equilibrium: Solution I

Two solutions are implemented to break strong equilibrium. Solution I is added realism in the form of structure rotation. As agents attach to a structure, and impart linear and rotational momentum the structure rotates about different axes. To simulate this effect, the structure is made to rotate about an arbitrary moment

$$\overrightarrow{R} = O_{rate}\ \widehat{[0.5,0.5},0.5] \qquad (1)$$

at a given rate

$$O_{rate} = \frac{\pi}{60}\frac{radians}{\text{sec}} \qquad (2)$$

A real structure rotates at a lower angular velocity as more satellites attach, and a corresponding change in the axis of rotation occurs over time. However, this model assumes that the structure maintains a constant O$_{rate}$ and $\overrightarrow{R}$. Continuous rate rotation is inherent without intervention, so a fixed angular velocity and angular moment is a viable option. In addition, the structure consists of motive satellites. Therefore, it is capable of station-keeping and maintaining a constant rotational velocity, and the model remains within the bounds of realism.

Rotation accomplishes one extremely important thing: a stirring action induced by rotating the structure. As the structure arcs through the constellation of agents, some are pushed aside. This action looks like a viscous fluid pouring over a solid object, and it increases the probability of assimilation to the structure from the constellation. Without stirring, agents tend toward a zero-velocity state. As the structure rotates through a field of agents, the agents are forced to take evasive action to avoid collisions with structural elements. In-path satellites have no choice but to move along a path parallel to $\overrightarrow{R}$. In doing so, they break equilibrium for a small amount of time and are more likely to spot active faces and seek assimilation. Problems remain even with *forced rotational mixing.* Satellites try to ride the forward wake of the rotating structure and refuse to flow gracefully out of the way. The result is unfortunate: due to this 'surfing' phenomena, a number of satellites remain in equilibrium, indefinitely.

5.1.6 Breaking Strong Equilibrium: Solution II

Solution II also adds realism. Additive system noise degrades the output of most systems. However, reality (noise) injected into the STEMS model solves more problems than it creates. In nature and in this implementation of distributed satellite behavior, a reasonable level of noise is required for annealing. This noise is varied to determine the optimal noise level. As the second parameter of interest in Chapter 5, *Results and Analysis*, noise levels are varied and we explore the effects of noise on a system of this type to explain the results. As mentioned earlier, thruster noise causes an uncertainty in position. Additive noise ensures that an agent never reaches an intended position exactly.

An analogy between gasses and nano-satellites is helpful at this point. Increased uncertainty in position for a satellites is analogous to increased temperature in a gas.

Also like a gas, satellites never stop moving, and system noise makes local minima settling less probable. How much less probable is a function of the variance of the additive system noise. If only a small amount of noise is added compared to the inter-satellite distance involved, then strong equilibrium is maintained and noise has less effect. Strong equilibrium is no longer a concern: 'surface tension' is broken and agents escape from equilibrium positions and transition to others. A satellite constellation can reach a *boiling point* with extreme dynamic noise. Such a boiling point is so excessive that the will of an agent no longer matters. Pro-activity is replaced with re-activity, and the mission fails (except by chance success). An upper bound (boiling point) is, therefore, imposed on the range of noise that an exhausitive analysis must explore. The lower bound is of this range is noise with zero variance. An optimal noise floor exists somewhere between these two bounds. If this noise floor is greater than the noise induced naturally, from the environment, then additional noise should be *injected* into the system for optimal performance.

However, the optimal noise variance depends on many other variables. Architectural style, the behavior function, and the available impulse power are all variables that determine the optimal noise floor. Four different styles of architecture are analyzed over two different noise magnitudes. To address the noise question, these eight simulations of 231 satellites over 350 frames were run for a period of two weeks. Ultimately, computational power limited the scope of this study. Simulations of 400+ satellites over 600 frames at 0.5 seconds per frame proved too large to analyze using STEMS. Results worthy of detailed study should cover 1,000,000+ satellites over 10,000 plus frames at millisecond intervals.

5.1.7 Results and Analysis Format

The battery of results presented below is applied to every case study in the following eight case studies. To handle the massive quantity of data generated by STEMS, a standard battery of plots is required. The analysis is organized in a concise manner beginning with case study A. The eight case study simulations took one week to run on an Air Force Institute of Technology (AFIT) Sparc Ultra

1. Visual snapshots of the time varying simulation illustrate with exceptional clarity the underlying behavioral mechanisms discussed in the *Overview*, namely, structural stirring, noise annealing, and structure formation. Two sets of four architectures (231 satellites, 350 frames) are analyzed at two separate noise variances. The results section concludes with a comprehensive analysis of the eight formation simulations. Each of the four architecture types are discussed individually and significant points made. Deductions are drawn that suggest correlations between system noise variance and group behavior. Finally, Chapter 6 concludes this document with the top 10% of all findings and future recommendations.

5.2 The Battery Defined

A set of plots is required to analyze the data. This battery of eight plots interprets the massive RUNN data structure that STEMS returns after a simulation of satellite collaborative behavior. The battery is designed to provide insight into the internal mechanism of this particular distributed system in the same way that projection onto Eigenvectors provides insight into higher dimensional clusters of points. Thus, the following plots generate interpretations that are not obvious from looking at the swarm illustrations alone. For example, they describe the mean inter-satellite distances over time and how far the net center of gravity deviates during structure formation. The result is a better understanding of the mechanisms that drive collaborative behavior. Finally, how to modify local behavioral rules to fit macro mission requirements becomes apparent.

5.2.1 Plot Type A

The first plot of the battery has three separate curves. Curve 1 is the joint Center of Gravity (CG^{sc}). It is the center of gravity of both the constellation agents and the structural agents and reveals how the entire system deviates from a linear path over time. For example, from case study A, the constellation deviates $\sim 8 meters$ from the origin over a period of 2 minutes 30 seconds – a deviation rate of 5.33 centimeters per second. The equation for the joint mean is

$$CG_i^{sc} = \sqrt{\left(\frac{1}{N^0}\sum_{j=1}^{N^0} P_x^{sc}\right)^2 + \left(\frac{1}{N^0}\sum_{j=1}^{N^0} P_y^{sc}\right)^2 + \left(\frac{1}{N^0}\sum_{j=1}^{N^0} P_z^{sc}\right)^2} \tag{3}$$

where $i : (0 < i \leq M)$ is the frame time index, N is the number of satellites in both the constellation and the structure, $P_{xyz} = \delta_j^p$ and sc implies structure/constellation agent positions. Curve 2 is the constellation CG^c and is determined in a similar manner

$$CG_i^{c} = \sqrt{\left(\frac{1}{N^0}\sum_{j=1}^{N^c} P_x^{c}\right)^2 + \left(\frac{1}{N^0}\sum_{j=1}^{N^c} P_y^{c}\right)^2 + \left(\frac{1}{N^0}\sum_{j=1}^{N^c} P_z^{c}\right)^2} \tag{4}$$

where c implies constellation agent positions only. The constellation CG_i^s is found in precisely the same manner. Results obtained in following sections suggest that the CG^{sc} experiences translation at a slow rate. Given the wide constellation distribution (several thousand meters at times) a slew rate of 5-10 centimeters per second is quite low. However, in a low earth orbit such a translation rate is capable of ending a mission or placing the finished structure in a different orbit than originally planned. The conclusion is that structural station-keeping *is* required to maintain orbit.

5.2.2 Plot Type B

The second plot of the battery has three different curves. Curve 1 is the number of structural satellites in the structure vs. time, Curve 2 is the number of satellites in the constellation vs. time, and Curve 3 illustrates the assimilation rate or the rate at which elements are converted from free-flying agents to structural members. The data structure RUNN contains a field STM of dimension [1 x N]. It contains the boolean values 0 and 1, where 1 signifies structural membership and 0 implies constellation membership. The number of structural elements is

$$E_i^s = \sum_{j=1}^{N} runn.f(i).stm_{1,j} : (1 \leq i \leq M) \quad (5)$$

where j is the standard index over the number of agents, e.g., $P_j = \delta_j^p$ and where P is the position of an agent in three dimensions. In all probability, $E_i^s = (N^0 - E_i^c)$, but we cannot assume this to be true always. If some agent δ_j were incapacitated or knocked entirely out of the constellation by space debris, then $E_i^s < (N^0 - E_i^c)$ and a separate summation must be calculated. The matrix STM does not contain a third state to describe incapacitation, and $E_i^s = (N^0 - E_i^c)$ is always considered to be true, i.e., agents are counted dead or alive. Curve 3 is determined by Equation (4)

$$Assimilation\_Rate_i^{c \rightarrow s} = runn.f(i+1).stm_{1,j} - runn.f(i).stm_{1,j} \quad (6)$$

where $(1 < i < M - 1)$. The assimilation rate is given in terms of agents assimilated per frame, not agents assimilated per second. A conversion is:

$$\frac{agents}{\text{sec}} = \frac{agents}{frame} \times \frac{frames}{\text{sec}} = Assimilation\_Rate_i^{c \rightarrow s} \times \frac{frames}{\text{sec}} \quad (7)$$

Paraboloid construction in Battery I demonstrates an assimilation rate of 2.16 $\frac{agents}{\text{sec}}$ from $\Gamma_{1.}$

5.2.3 Plot Type C

The third plot in the battery has two curves. Curve 1 illustrates the mean magnitude of swarming satellite tangential velocity ($\delta_j^T \; \forall \; \{0 < j < N^c\}$), where N^c is the cardinality of the set (β^c)of fleet satellites. Curve 2 is the standard deviation (STDEV) of this magnitude over the set of all swarming agents at a given time (the set of all constellation satellites is sometimes referred to as β^c). Plot generation becomes exponentially time consuming as the number of satellites in either β^c or β^s

increases. The simulation data stored in RUNN for each of the eight case studies consumed 360 Megabytes of data and was broken into two separate data structures, SRUNN 'structural run file' and CRUNN 'constellation runn file.' These two new data structures contain ephemerides for β^s and β^c respectively instead of jointly as with RUNN and make it easier to analyze properties of the structure or swarm independently. Curve 1 is generated by substituting

$$\left\|T_j^c\right\| = \sqrt{(T_{j,x}^c)^2 + (T_{j,y}^c)^2 + (T_{j,z}^c)^2} \tag{8}$$

in

$$MSV_i^c = \frac{1}{N}\sum_{j=1}^{N}\left\|T_j^c\right\| \tag{9}$$

and using the relation

$$\sigma_i = \sqrt{E[x^2] - (E[x])^2} \tag{10}$$

where, x is given by $\left\|T_j^c\right\|$, to obtain

$$\sigma_i = \sqrt{\frac{1}{N}\sum_{j=1}^{N} T_j^c - \left(\frac{1}{N}\sum_{j=1}^{N}\left\|T_j^c\right\|\right)^2} \tag{11}$$

Thus Curve 2 is the root central moment of satellite velocity over β^c. Trends indicate that satellite velocities are greatest during the payload to equilibrium transition Δ_{01} and can fluctuate rapidly when structure formation is initialized by δ_o. For example, case study A yields mean velocities ranged from 0.2 to 0.7 m/s with a standard deviation near 0.125 m/s. This result suggests that every satellite in the swarm tends to travel at approximately the same velocity after equilibrium is reached. Agent velocities become particularly alike when transitions affecting the fleet as a whole are initialized. When δ_o turns on active faces, the standard deviation of satellite velocity over all agents in the fleet decreases. However, this result depends strongly upon the system noise floor, style of architecture, and structure rotation rate.

5.2.4 Plot type D

The fourth plot in the battery has two curves. The first curve represents the mean of constellation thruster impulse magnitude in Newton seconds (kg·m/s). The second curve represents the standard deviation of this impulse magnitude. Both curves are calculated in the same manner as the tangential velocities of plot type C, except the variables change to a separate (N^c x 3) column in the CRUNN structure, *crunn.f(i).IT*. The CRUNN 'constellation runn structure' is employed as a representation of β^c. The analysis using impulse thruster magnitude instead of satellite velocity may be visualized using

$$\left\|I_j^c\right\| = \sqrt{(I_{j,x}^c)^2 + (I_{j,y}^c)^2 + (I_{j,z}^c)^2} \tag{12}$$

which yields the Impulse magnitude for some satellite δ_j. using

$$MSI_i^c = \frac{1}{N}\sum_{j=1}^{N} \left\|I_j^c\right\| \tag{13}$$

for Curve 1 and using

$$\sigma_i = \sqrt{E[x^2] - (MSI_i^c)^2} \tag{14}$$

where x is given by $\left\|T_j^c\right\|$ gives

$$\sigma_i = \sqrt{\frac{1}{N}\sum_{j=1}^{N^c} I_j^c - (\frac{1}{N}\sum_{j=1}^{N^c} \left\|I_j^c\right\|)^2} \tag{15}$$

which yields Curve 2, the standard deviation of satellite thruster impulse magnitude over β^c. Plots of MSI^c best define states $\Gamma_{0,1,2}$ with sharp changes in mean impulse output constellation wide. A large spike in impulse is noticed when $t = 0.5$ for the payload break and again when δ_0^c transitions to δ_0^s. Finally, a third spike occurs from the assimilation frenzy rebound early in the Δ_{23} transition period. The net impulse expended for the entire construction is computed using

$$Net\_I^c = \sum_{i=1}^{M} N_i^c MSI_i^c \tag{16}$$

and the average impulse expended per satellite at any given synchronous burn is obtained by solving:

$$\delta^{impulse} = \frac{Net\_I^c}{\sum_{i=1}^{M} N_i^c} \tag{17}$$

However, this procedure predicts the actions of an average satellite in the constellation. Satellites on the *outer rim* of the fleet or in the *center* expend different quantities of fuel over time due to the average distance they must travel before assimilation. Additive system noise also contributes to the average quantity of fuel spent during the construction act.

5.2.5 Plot Type E

The fifth plot in the battery also consists of two curves. The first curve illustrates the Mean Magnitude of Net Received Power on Line of Sight Channel-0 over N^c, and the second curve is the standard deviation. To understand Curve 1, imagine a single satellite in orbit. The line-of-sight sensors on this satellite are represented by unit vectors normal to each of its six faces. The component of power received is stored in matrix **RP0** of dimension (1 x 6). The elements of **RP0** are stored as magnitudes that refer to $\{+X, -X, +Y, -Y, +Z, -Z\}$ unit vectors, respectively. Summing the elements of **RP0** returns the net received power for some agent. Let us revisit the

SRUNN 'Structural data structure' that MATLAB® supports in versions ≥ 5.0. If we address *srunn.f(i).RP0*$_{j,:}$ **RP0** of dimension (1 x 6) is returned and we may compute:

$$P\_received_{i,j} \;=\; \sum_{k=1}^{6} \mathbf{RP0}_{j,k} \qquad (18)$$

To generate curve one, we take the average over all agents in β^c :

$$MRP0_i^c \;=\; \sum_{j=1}^{N^c} P\_received_{i,j} \qquad (19)$$

Thus Curve 2, the standard deviation of P_received over j, is found by typing STDEV (p_received) in MATLAB® or by referring to Equation (14). This plot is capped at 100 mW of received power to leave room for more interesting received power figures in later frames. When two satellites are docked in payload, the net received power is at a maximum of 600 mW or 100 mW on a side. Received power decreases proportional to the inverse square of the distance and reaches a minimum mean magnitude and variance in the equilibrium position (maximum inter-satellite distance). Results demonstrate a power plateau during the major construction phase caused by the balance between structural attractive forces. This plateau can be raised or lowered by modifying the MOOD0.M or MOOD1.M MATLAB® functions.

5.2.6 Plot Type F

The sixth plot in the battery has two curves. The first represents the Mean Received Power on Channel-1 vs. Time, instead of Channel-0. Recall that Channel-0 deals with swarming and structural formation social behavior, while Channel-1 causes overwhelming attraction to active structural faces. The second curve of plot type F represents the standard deviation of the received power over β^c. Both curves are calculated in the same manner as the those of plot type E, except the variables change to a separate (N^c x 6) column in the CRUNN structure, or c*runn.f(i).RP1* (refer to the previous subsection and replace instances of **RP0** with **RP1**). As a check, results generated for Curve 1 and 2 suggest that no power is received from the structure when it is not present. Furthermore, the received power begins low and peaks at the time that satellite velocities are lowest. At the peak a battle between social Channel-0 and Channel-1 begins. This battle causes the satellite assimilation rate to decrease, and structure formation effectively stops. Notice that satellites remain at the end of every simulation. Eventually, active faces draw them in, but not before debilitating memory swapping begins and the simulation files become too large to manipulate in MATLAB®. The beauty of this system is that the real world implementation is far simpler than the software simulation because this one model is simulating a massively parallel problem, but with a serial computer.

5.2.7 Plot Type G

The seventh plot in the battery has two curves. The first curve represents the Mean Inter-Satellite Distance vs. Time and the second curve represents the Mean Position Relative to the origin [0,0,0]. The number of inter-satellite distances that must be calculated per satellite in N^c is

$$NISD = \frac{N^c(N^c - 1)}{2} \tag{20}$$

and hence the computation intensive portion of this work appears: a polynomial dependance on the number of satellites being simulated. To generate Curve 1, the elements of a symmetric inter-satellite distance matrix **D** are summed and divided by $NISD$

$$MID_i^c = \sum_{x=1}^{N^c-1} \sum_{y=x}^{N^c} \| P^c(x,y) - P^c(y,x) \| \tag{21}$$

is computed, where P is an (N^c x 3) matrix or *crunn.f(i).P.* Similarly, Curve 2 is calculated by substituting the origin

$$MIDO_i^c \;=\; \sum_{x=1}^{N^c-1} \sum_{y=x}^{N^c} \|P^c(x,y) - [0,0,0]\| \tag{22}$$

expected, Equation (21) produces the roughly the same trend as Equation (22). However, Equation (22) assumes that the swarm tends about the origin, as most results generated with *plot type A* indicate. Trends for C-Agent (Constellation Agents) MID^c and $MIDO^c$ demonstrate an increase in inter-satellite distance after payload break. The mean inter-satellite distance at equilibrium for the *Cylinder*, Case Study A, run is 28 meters, suggesting that any given satellite is 28 meters from any other on average (mean inter-satellite distance should not be confused with the mean distance to local neighbors; it is considerably lower in magnitude). As the average received power on both Channels 0 and 1 tend toward a steady state, MID^c and $MIDO^c$ level off and the satellites tend to remain evenly spaced (until $N^c = 0$, of course).

5.3 The File System

The Structural Emergence Simulator (STEMS) is a third generation Graphical User Interface (GUI). Certain functions were optimized for speed, namely; MCAD2.M, which rotates bodies about an arbitrary axis in three-dimensions. To run simulations on a Unix or PC platform between versions of MATLAB® (5.0, 5.1, and 5.2),

NIKITA.M *internal variables structure*		
Variables	Value	Description
Payload_Radius	4	Approximate radius of a radial payload
Payload_Height	7	Exact height of a payload in cube widths
NumberofFrames(M)	300	The number of frames being simulated (at 0.5 $\frac{sec}{frame}$)
Γ_1^i *seed*	60	Frame upon which to initialize δ^0

Table 5.2 NIKITA.M internal variables for NIGHTHAWK engine

a separate engine was extracted from STEMS. This engine is known as NIGHTHAWK and runs simulations based upon two functions: NIKITA.M and BLADERUNNER.M. Nikita describes a structure with the variables in Table (2) and passes them to BLADERUNNER.M for overnight or week-long simulations. Using the NIGHTHAWK engine, a process termed DEVIL47-300CYL.M was started on an AFIT Unix machine. Four-hundred satellites are simulated over 300 Frames in this battery; 210 out of 400 satellites attached to the structure by frame 300 and the majority, $\Gamma_0 \rightarrow \Gamma_2$, behavior states are demonstrated.

Each frame in this simulation represents 0.5 real-world seconds for a total of 150 seconds (2 minutes 30 seconds) from start to finish. The style of architecture in is a flattened paraboloid. A single valued function that describes the surface in three dimensions is:

$$Z(X,Y) = \frac{1}{10}(X^2 + Y^2) \tag{23}$$

The complexity of $Z(X,Y)$ is of no consequence to the agents. In other words agents never failed to construct a structure based upon the complexity of the single valued function used to describe the structure.

One example blueprint is Equation (23). A simple equation is chosen to help visualize the result in three dimensions. In Γ_0 agents exist in an initial solid state, (a payload).

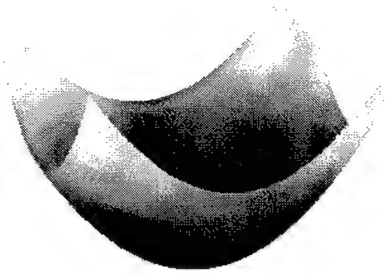


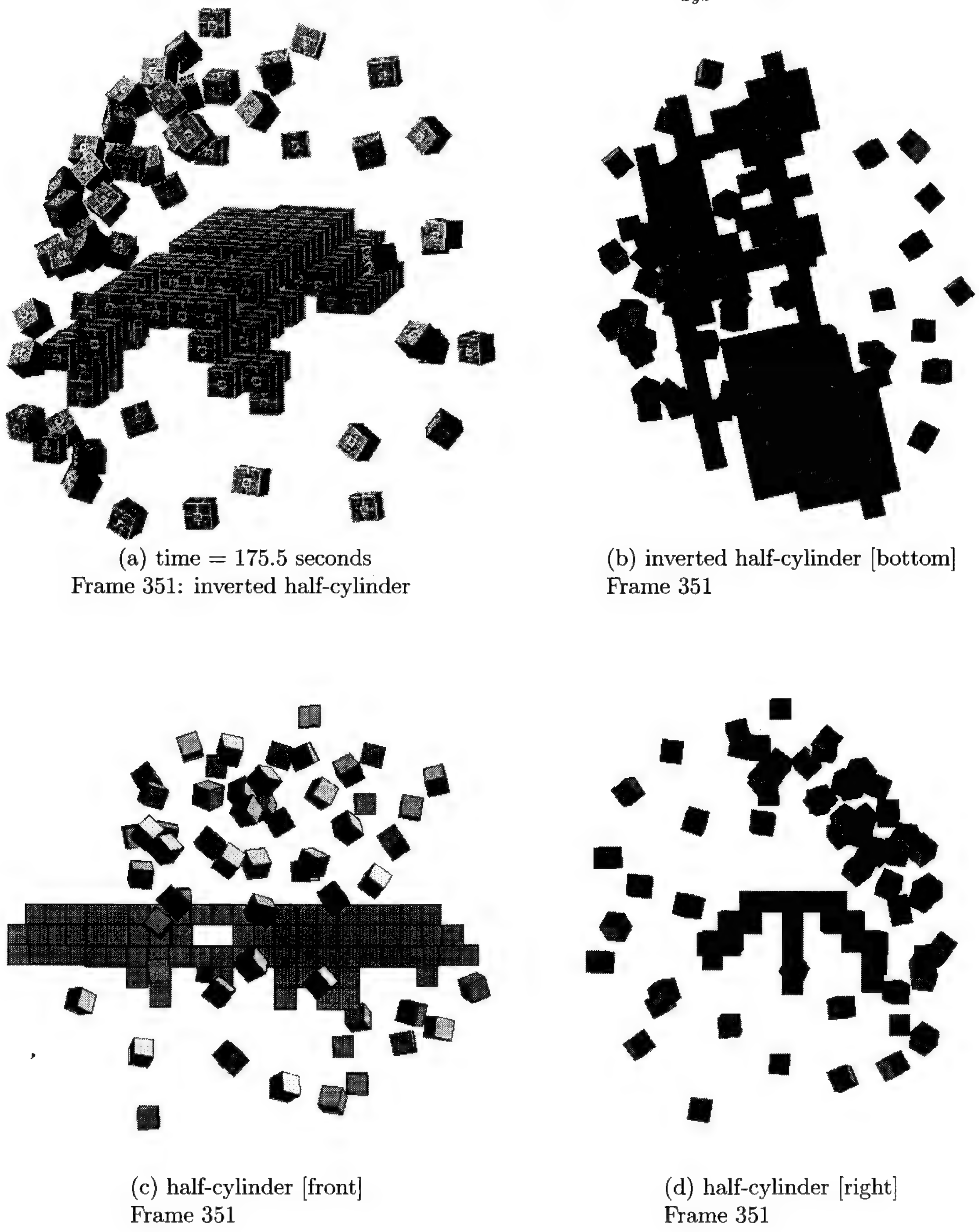
Figure 5.1 $Z(X,Y) = \frac{1}{10}(X^2 + Y^2)$

5.4 Eight Architecture Style vs. System Noise Case Studies

5.4.1 Description

The eight case studies presented on the following pages are two-dimensional representations of a three-dimensional system. Instances in time are presented to convey the time-varying nature of the results. Unfortunately, a color copy is not *explicitly* required, so the results are presented in black and white. Movies are available upon request.

Case Study A: *351 frames, 231 satellites:* *System noise* $\sigma^2_{xyz} = 0.0025$



(a) time = 175.5 seconds
Frame 351: inverted half-cylinder

(b) inverted half-cylinder [bottom]
Frame 351

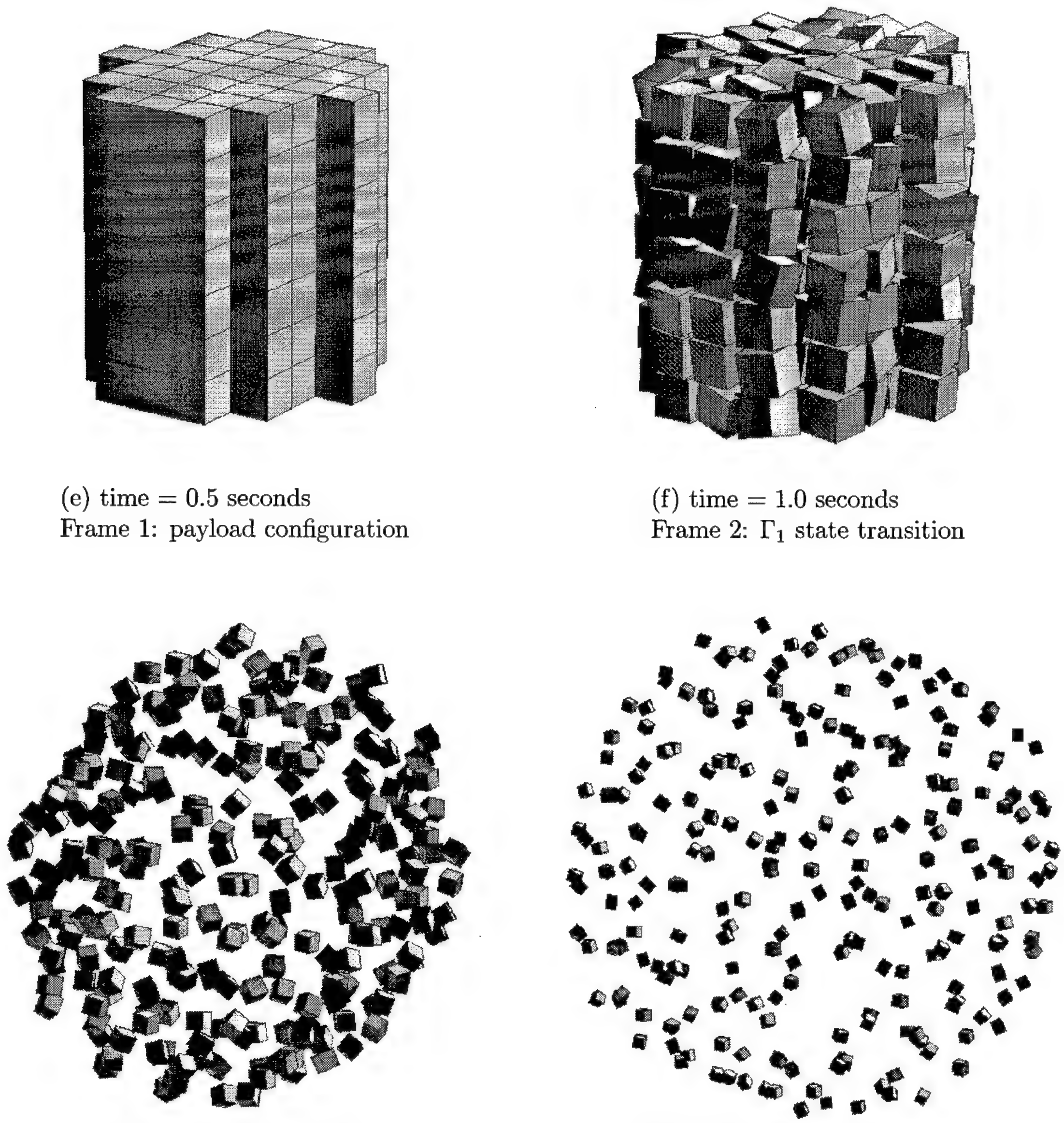
(c) half-cylinder [front]
Frame 351

(d) half-cylinder [right]
Frame 351

Snapshot at frame 351 with gray, low-resolution texture mapping [upper left]; (c) *Illustration of inverted half cylinder looking at the XY plane* [upper right]; (d) XZ plane [lower left]; *YZ plane* [lower right] Comment: notice the vertical column in image (d). The first structural agent is at the bottom of this column. Positive Z faces are switched to channel-1 (structural force) until construction according to the blueprints is feesible.

Figure 5.2 [a,b,c,d] Inverted Half-Cylinder constructed with low system noise.

Case Study A [page 2]: *351 frames, 231 satellites* *System noise* $\sigma^2_{xyz} = 0.0025$



(e) time = 0.5 seconds
Frame 1: payload configuration

(f) time = 1.0 seconds
Frame 2: Γ_1 state transition

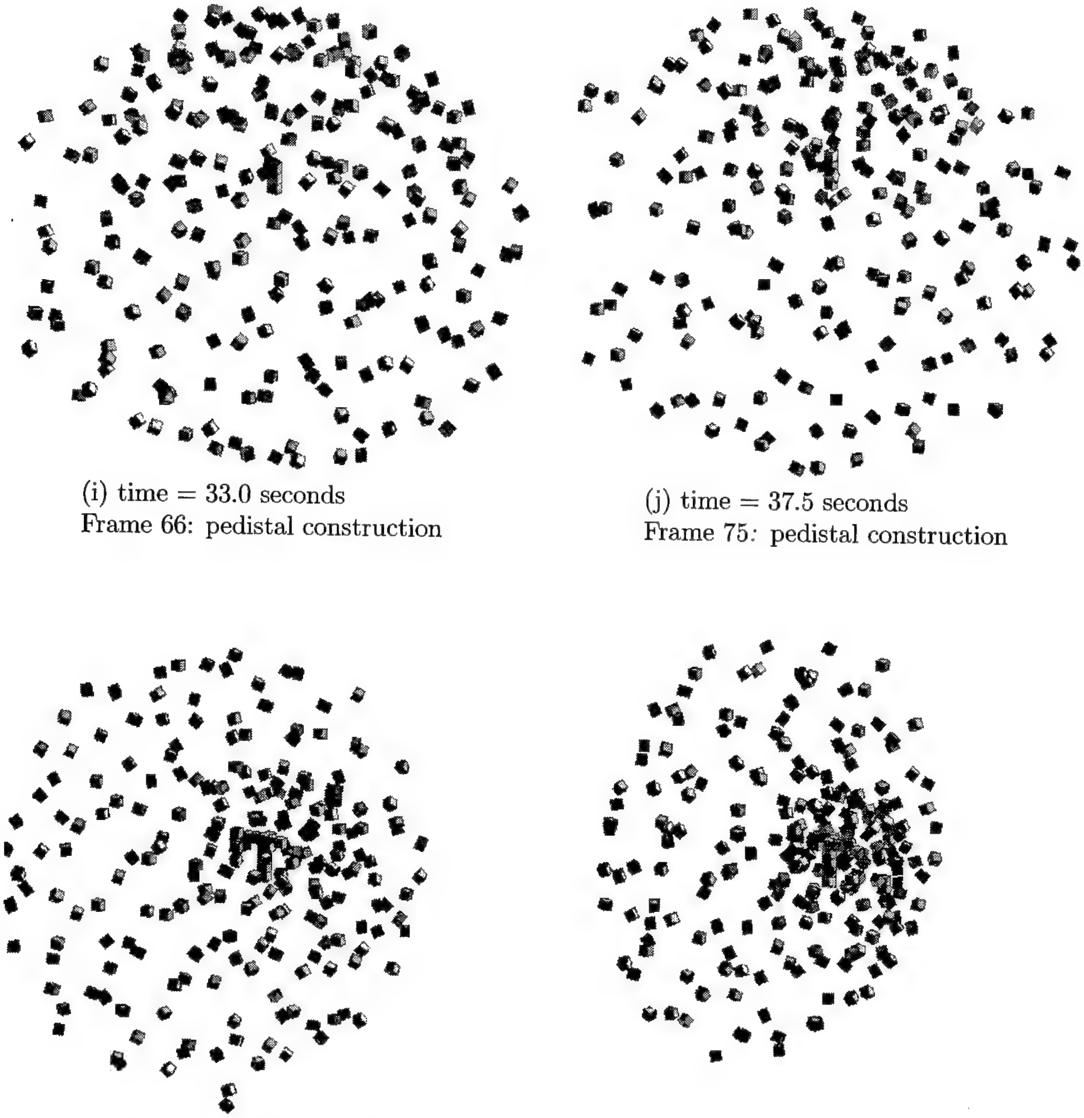
(g) time = 10.0 seconds
Frame 20: early Δ_{12}

(h) time = 25 seconds
Frame 50: Γ_2 state transition

Comment: Satellite impulse firing reaches the maximum allowable during the initial payload break. On-board sensors detect an impending collision and suggest excape. Since satellites in the center of the payload receive equal power from all sides, thruster are not fired. The result is a ripple effect. Thus, the payload sheds uniform sheets of satellites until dissolved. This is the constellation's first example of emergent behavior.

Figure 5.3 [e,f,g,h] Payload to equilibrium transition with low dynamics noise.

Case Study A [page 3]: *351 frames, 231 satellites* *System noise* $\sigma^2_{xyz} = 0.0025$



(i) time = 33.0 seconds
Frame 66: pedistal construction

(j) time = 37.5 seconds
Frame 75*:* pedistal construction

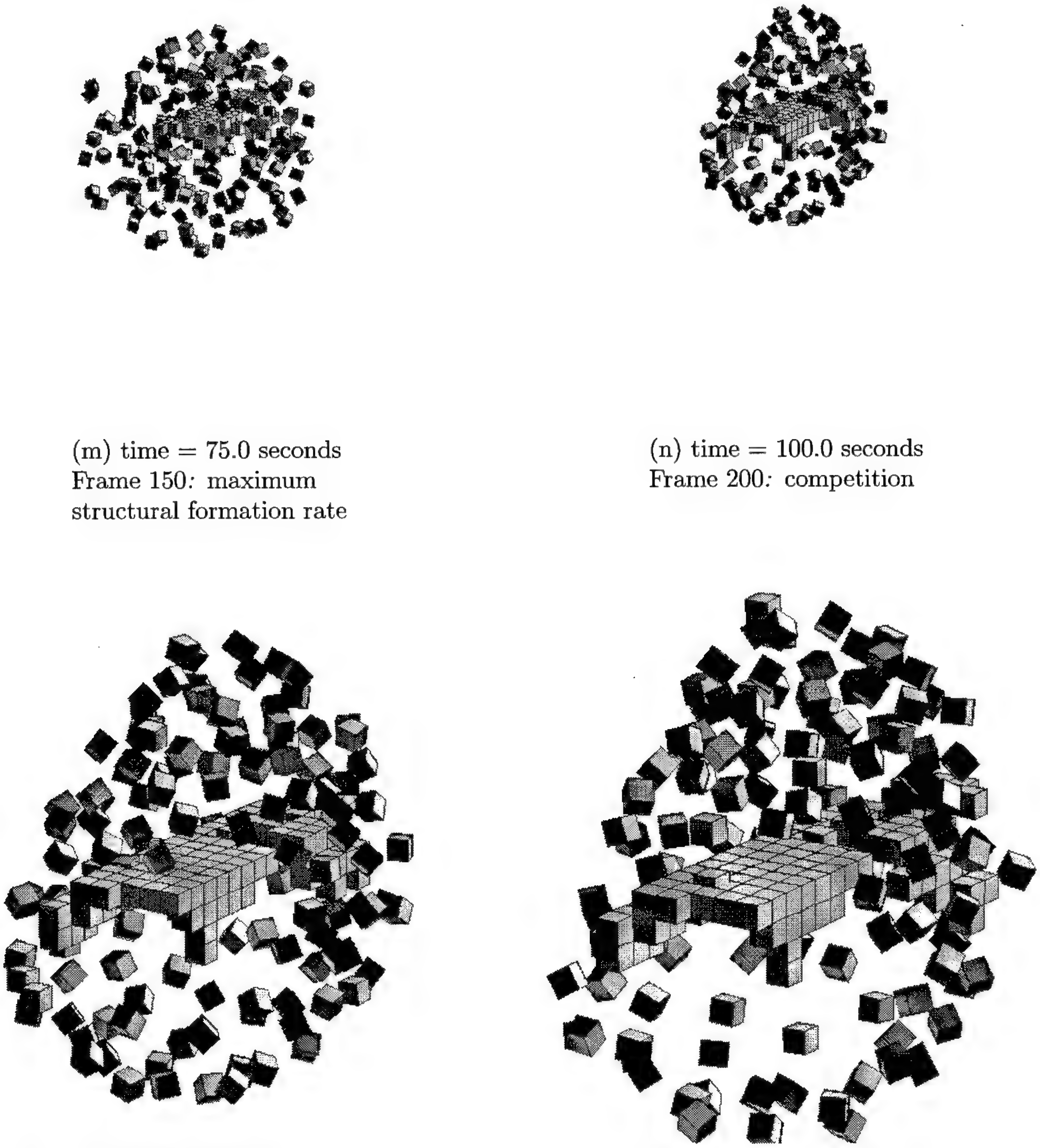
(k) time = 45.0 seconds
Frame 90: half-cylinder construction begins

(l) time = 25 seconds
Frame 100*:* construction rate increases

Comment*:* The satellite with ID-0 (δ^0) flys to the constellation center and activates. Surrounding satellites are attracted and attach to the structure. Each new satellite is passed local information in the form of structural blueprints and position. Frame (d) demonstrates both the effect of structural rotation and of the polynomial increase in active structural satellites; evident by the dense central cluster.

Figure 5.4 [i,j,k,l] Structure formation begins and construction activity peaks.

Case Study A [page 4]: *351 frames, 231 satellites* *System noise* $\sigma^2_{xyz} = 0.0025$



(m) time = 75.0 seconds
Frame 150: maximum structural formation rate

(n) time = 100.0 seconds
Frame 200: competition

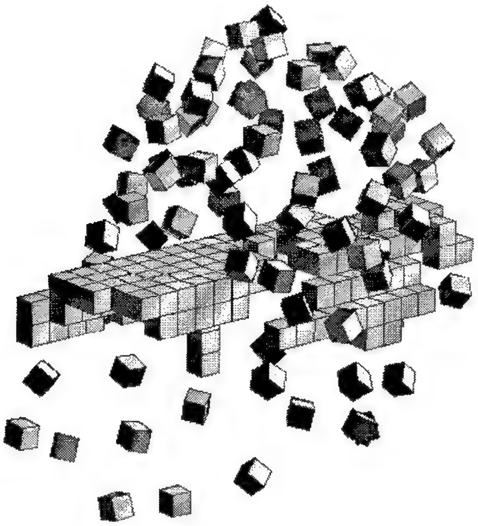
(o) time = 100.0 seconds
Frame 200: 100 meter perspective

(p) time = 105.0 seconds
Frame 210: quiescent state

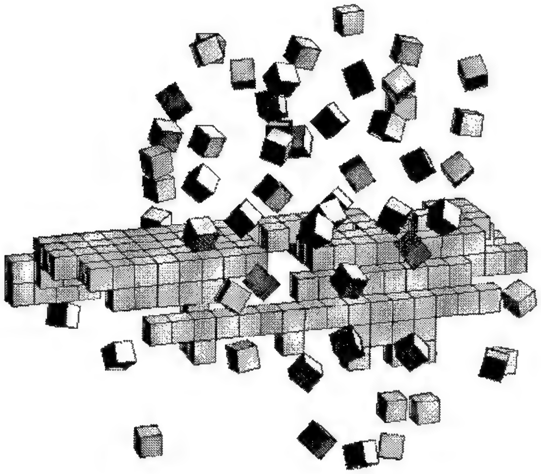
Comments: STEMS allows multiple satellite assimilations per frame. This feature is true to reality, in which thousands (or millions) of structural elements may connect asynchronously within a short period of time. Multiple satellites often compete for the same active satellite and for the same set of active faces. Assimilation scheduling is a computation intensive task.

Figure 5.5 [m,n,o,p] Post half-life construction rate approaches an asymptote.

Case Study A [page 5]*: 351 frames, 231 satellite constellation with system noise* $\sigma_{xyz}^2 = 0.0025$



(q) time = 2 minutes and 5.0 seconds
Frame 250*:* maximum structural formation rate

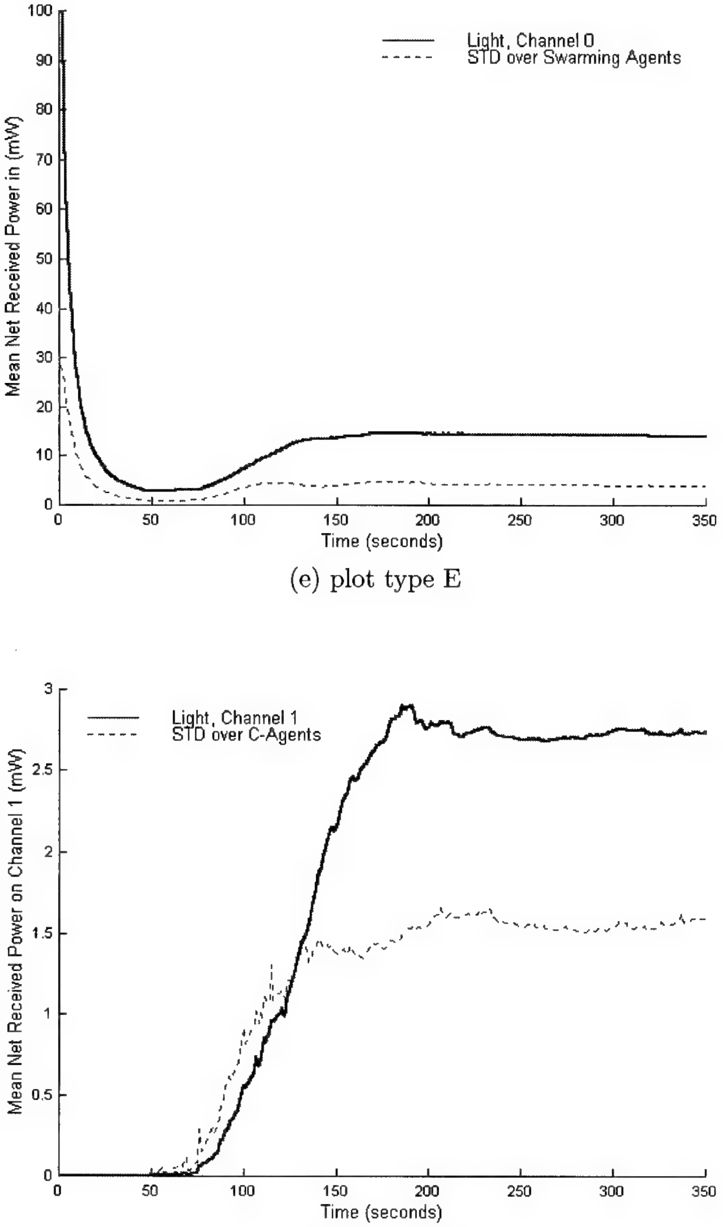


(r) time = 2 minutes 55.5 seconds
Frame 351 of 351*:* final frame

Comments*:* The following battery of plots points out one of the unique problems encountered nano-satellite structure formation. The assimilation rate drops to zero before every satellite in the constellation becomes a structural member. The result is a cluster of satellites in equilibrium about a slowly rotating structure. The problem, as mentioned in the Chapter III, Methodology, is caused by structures that rotate about a stationary axis only.

Figure 5.6 [q,r] Construction rate as a function of structural rotation

Case Study A*: Plots five and six of eight, Tranceiver CH-0 and CH-1.*

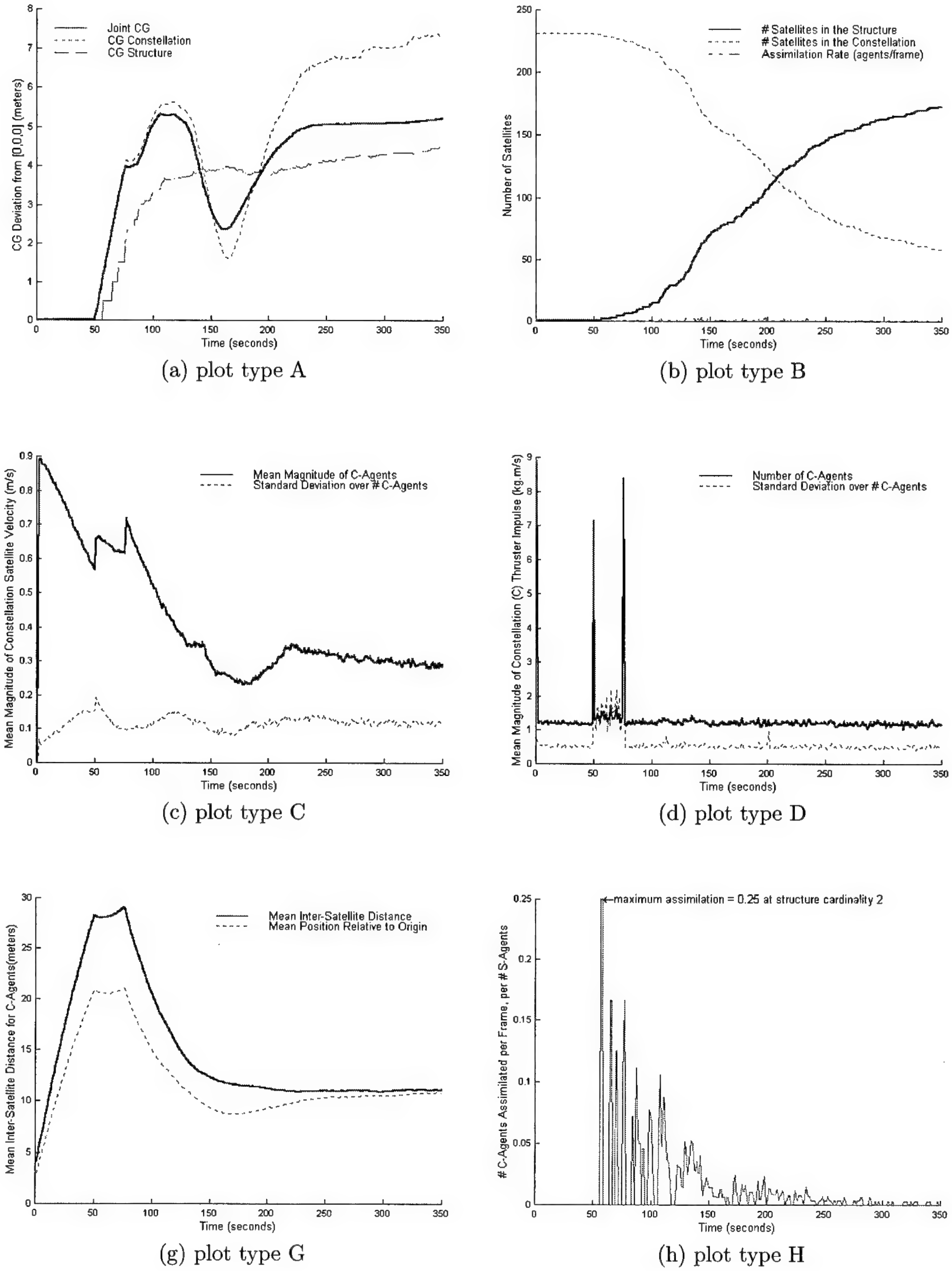


(e) plot type E

(f) plot type F

Figure 5.7 [E,F] Channel 0 and Channel 1 received power as a function of time

Case Study A: *plots 1,2,3,4,7, and 8 of 8* *System noise* $\sigma^2_{xyz} = 0.0025$



(a) plot type A

(b) plot type B

(c) plot type C

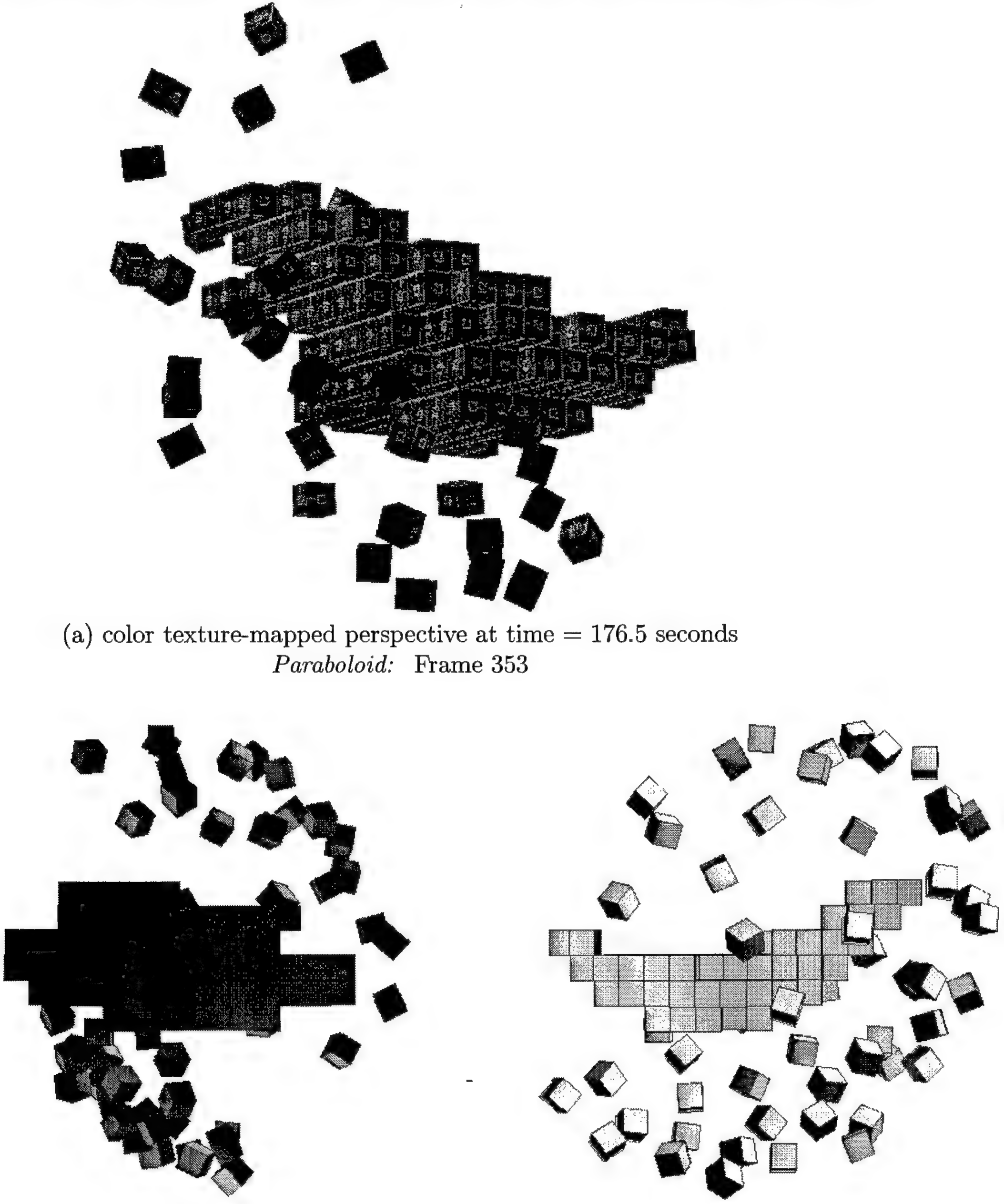
(d) plot type D

(g) plot type G

(h) plot type H

Figure 5.8 Battery A plots [a,b,c,d,g,h]

Case Study B*: 353 frames, 231 satellite constellation with system noise* $\sigma_{xyz}^2 = 0.0025$



(a) color texture-mapped perspective at time = 176.5 seconds
Paraboloid: Frame 353

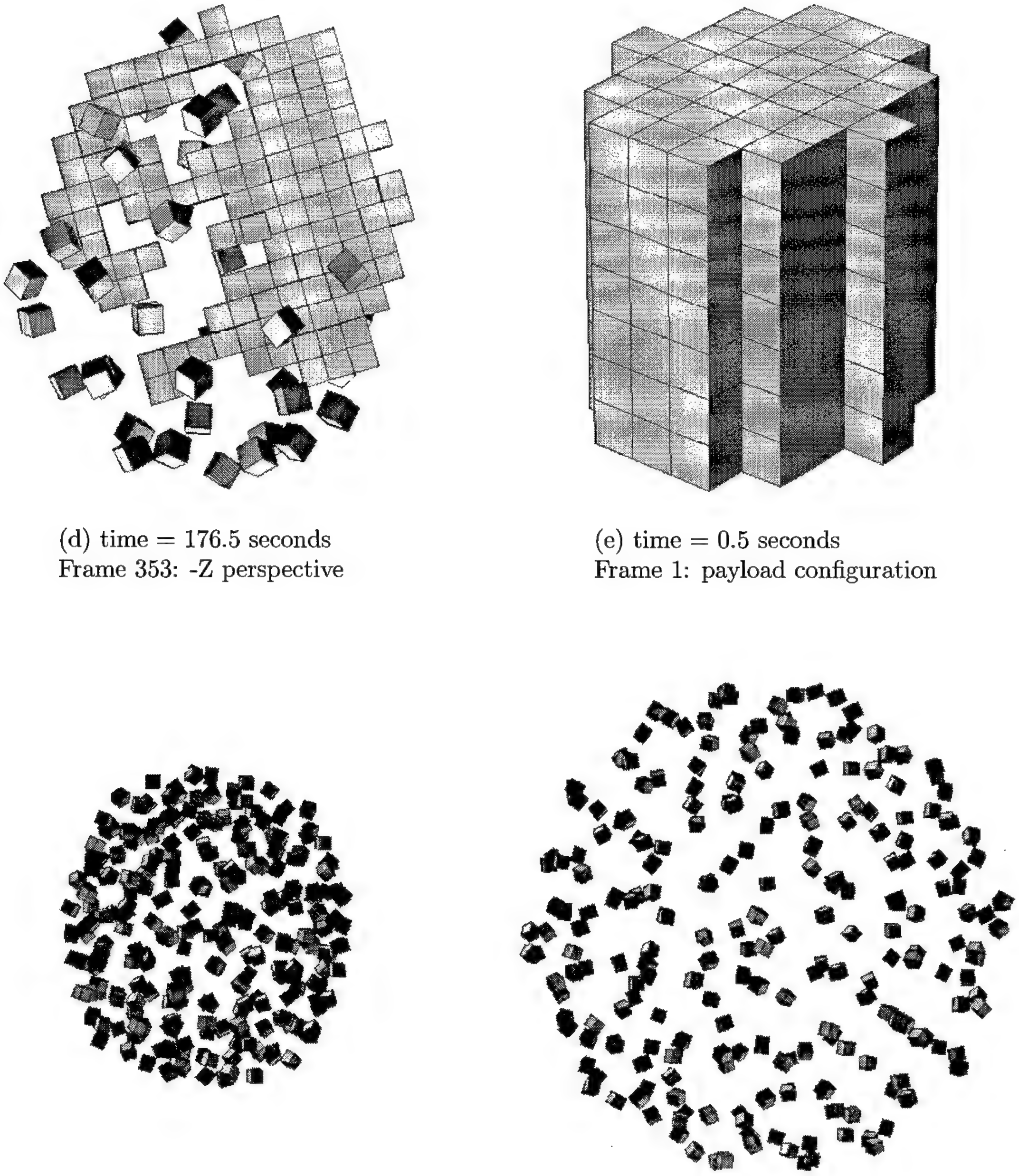
(b) *Paraboloid* [left]
Frame 353

(c) *Paraboloid* [front]
Frame 353

Figure (c) best illustrates paraboloid construction. Notice that the right side on the front view is at a more advanced stage of construction than the left. Knowledge is not passed between satellites; therefore, incoherent structure formation is unavoidable Several solutions to this problem exist. First, the structure formation algorithm may implement a death scenario. Second, uneven structure rotation increases homogeneity.

Figure 5.9 [a,b,c] paraboloid constructed with low system noise,.

Case Study B*: [*page 2*] 353 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0025$



(d) time = 176.5 seconds
Frame 353: -Z perspective

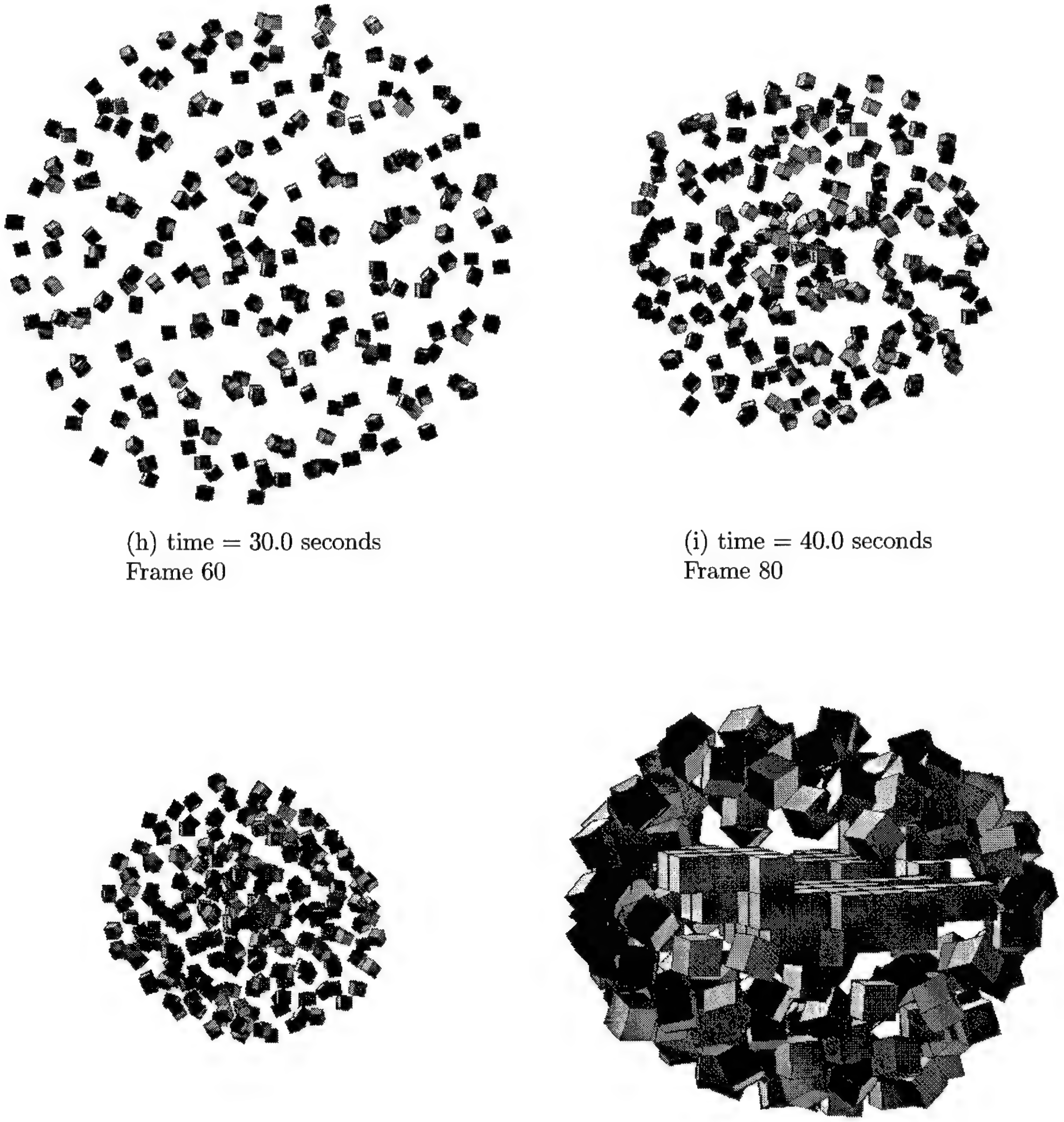
(e) time = 0.5 seconds
Frame 1: payload configuration

(f) time = 10.0 seconds
Frame 20: 40% of Δ_{12} complete

(g) time = 20.0 seconds
Frame 40: 80% of Δ_{12} complete

Figure 5.10 [d,e,f,g] Radial payload burst

Case Study B*: [*page 3*] 353 frames, 231 satellite constellation with system noise* $\sigma_{xyz}^2 = 0.0025$



(h) time = 30.0 seconds
Frame 60

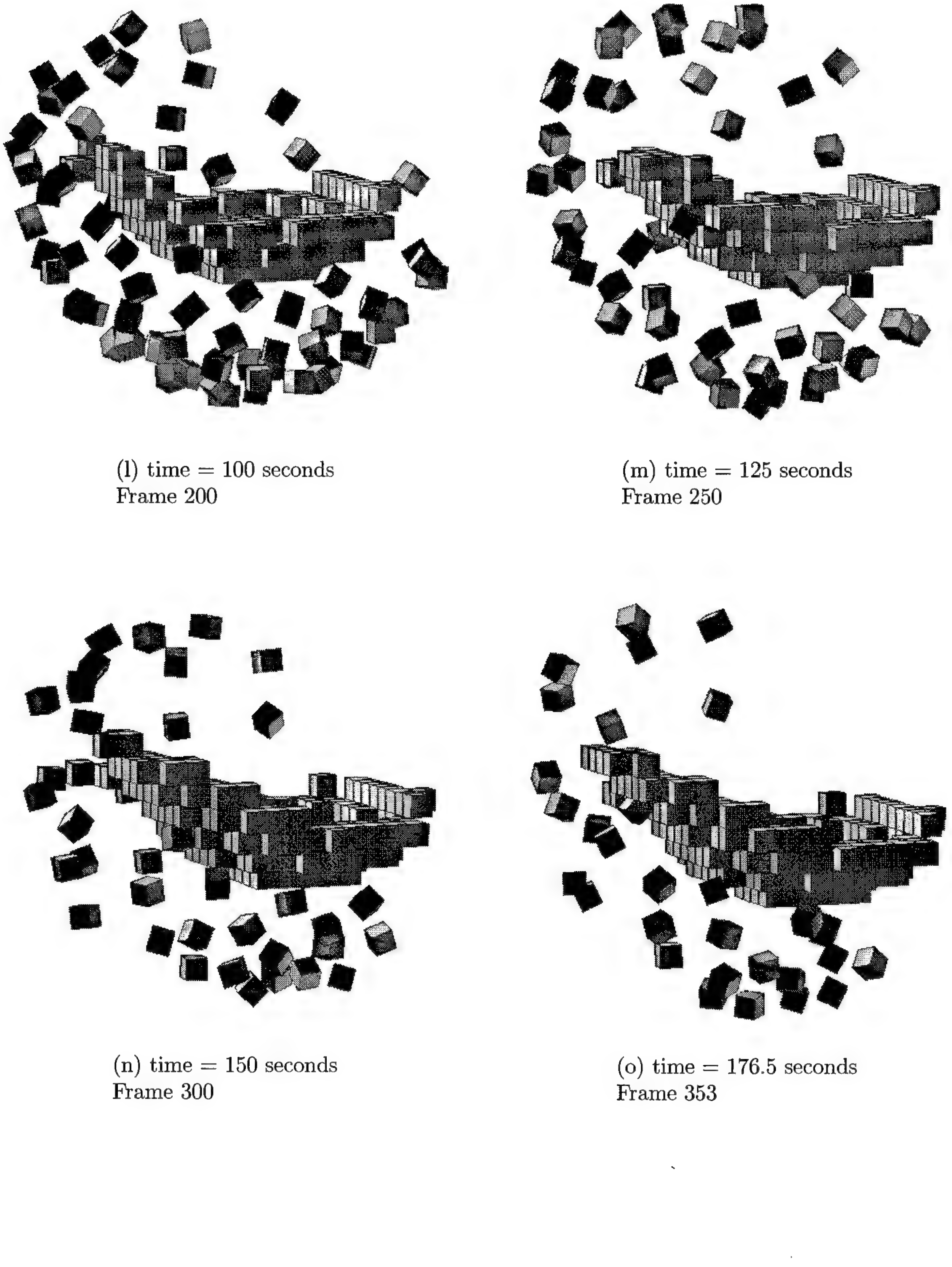
(i) time = 40.0 seconds
Frame 80

(j) time = 50.0 seconds
Frame 100

(k) time = 75.0 seconds
Frame 150

Figure 5.11 [h,i,j,k] Collapse from equilibrium and a peeling example

Case Study B: *[page 4] 353 frames, 231 satellite constellation with system noise* $\sigma_{xyz}^2 = 0.0025$



(l) time = 100 seconds
Frame 200

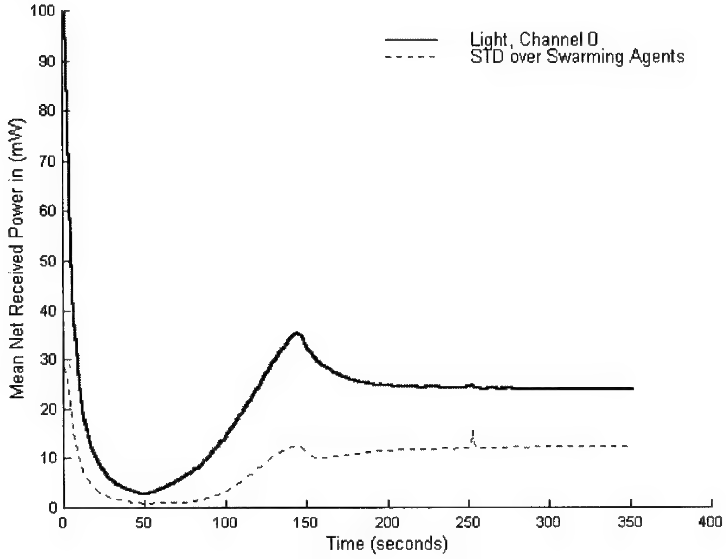
(m) time = 125 seconds
Frame 250

(n) time = 150 seconds
Frame 300

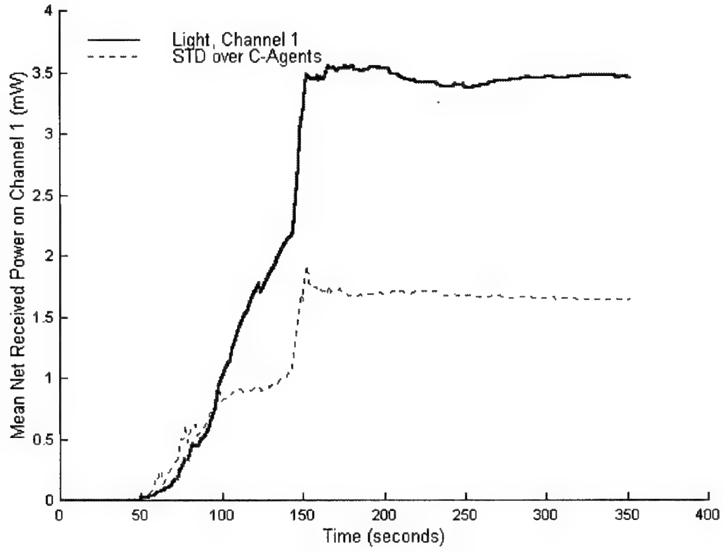
(o) time = 176.5 seconds
Frame 353

Figure 5.12 [l,m,n,o] The final stages of paraboloid construction

Case Study B*: Plots five and six of eight, Tranceiver CH-0 and CH-1.*



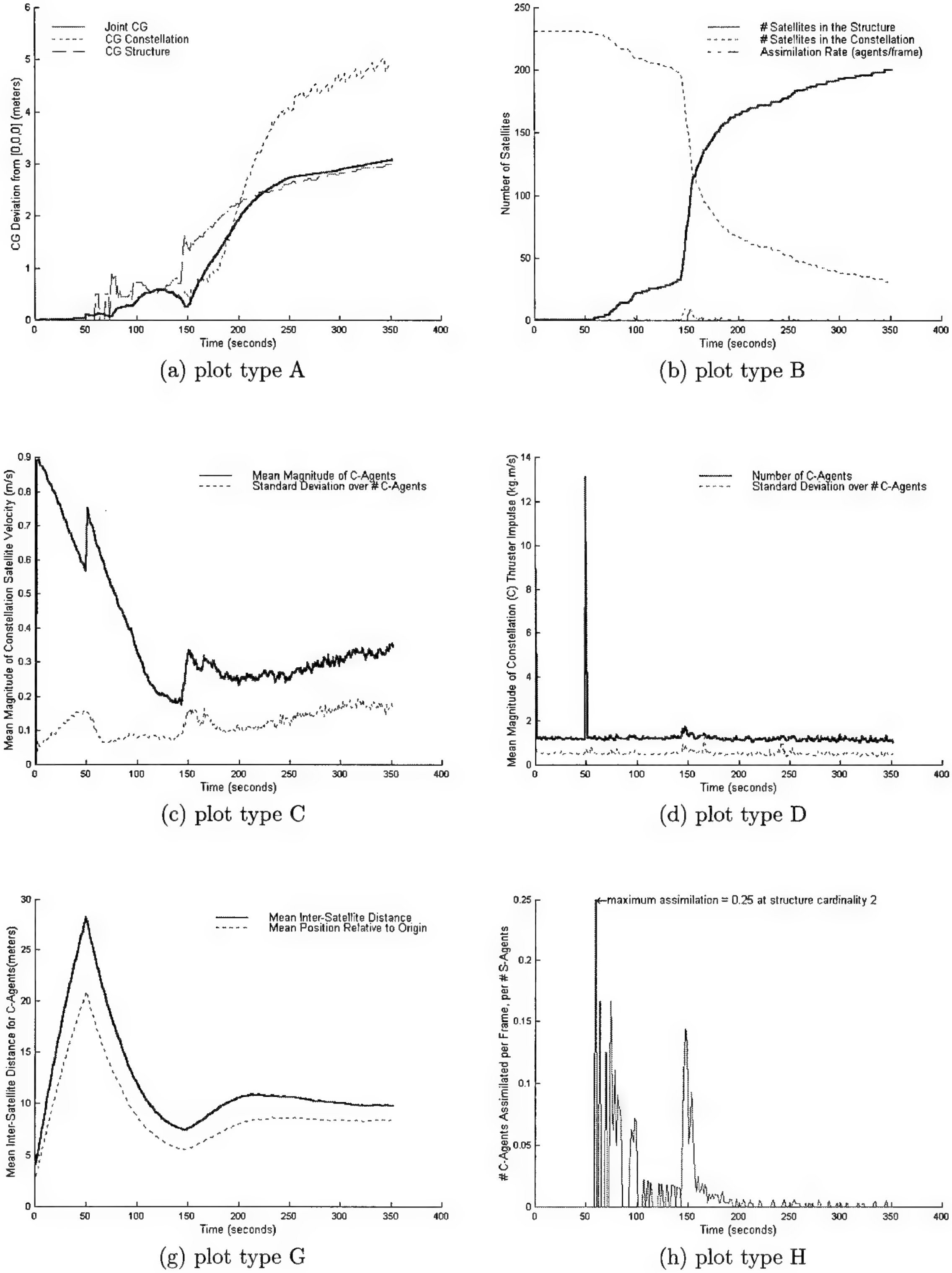
(e) plot type E



(f) plot type F

Figure 5.13 [E,F] CH-0 and CH-1 received power as a function of time

Battery B: *Plots 1,2,3,4,7, and 8 of 8* *System Noise* $\sigma^2_{xyz} = 0.0025$



(a) plot type A

(b) plot type B

(c) plot type C

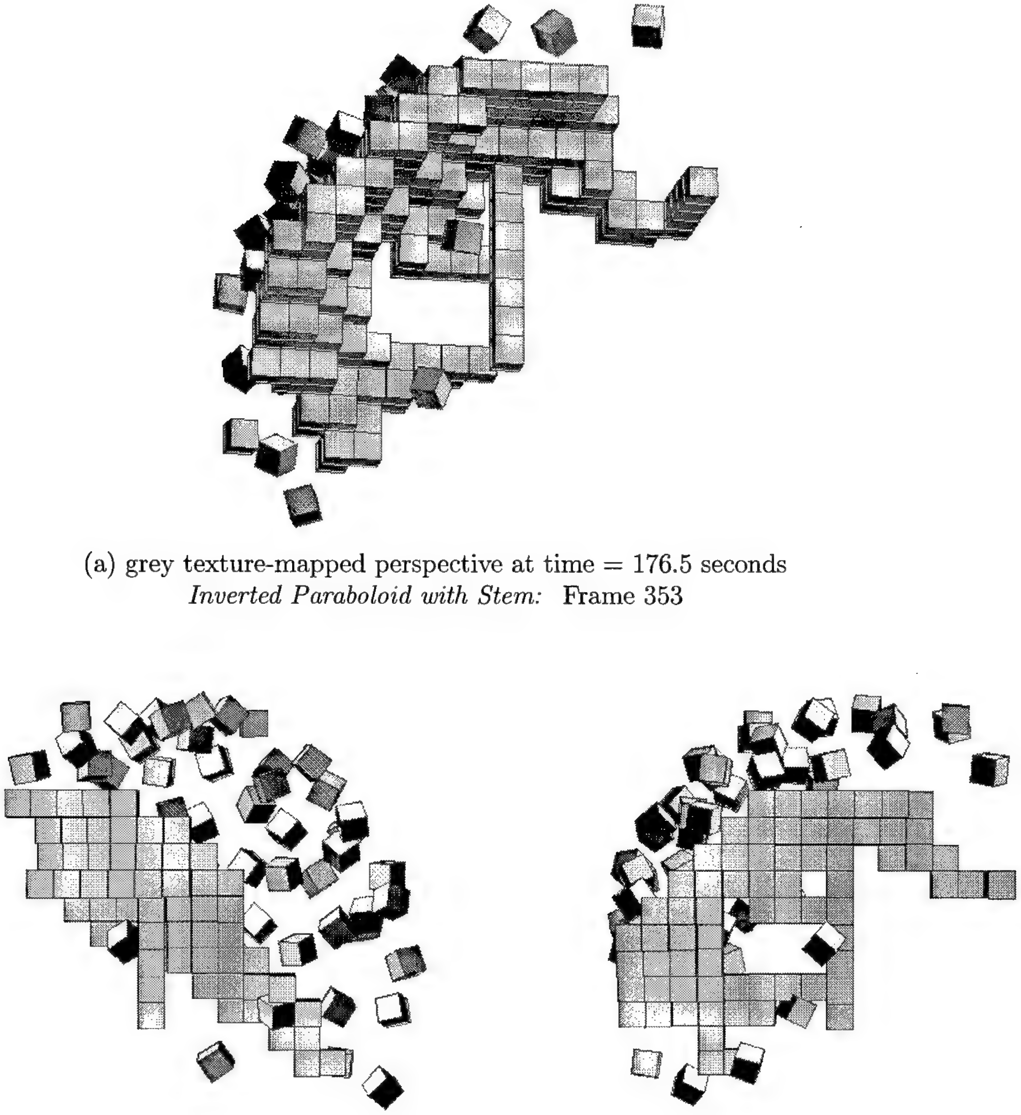
(d) plot type D

(g) plot type G

(h) plot type H

Figure 5.14 Battery B plots [a,b,c,d,g,h]

Case Study C*: 354 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0025$



(a) grey texture-mapped perspective at time = 176.5 seconds
Inverted Paraboloid with Stem: Frame 353

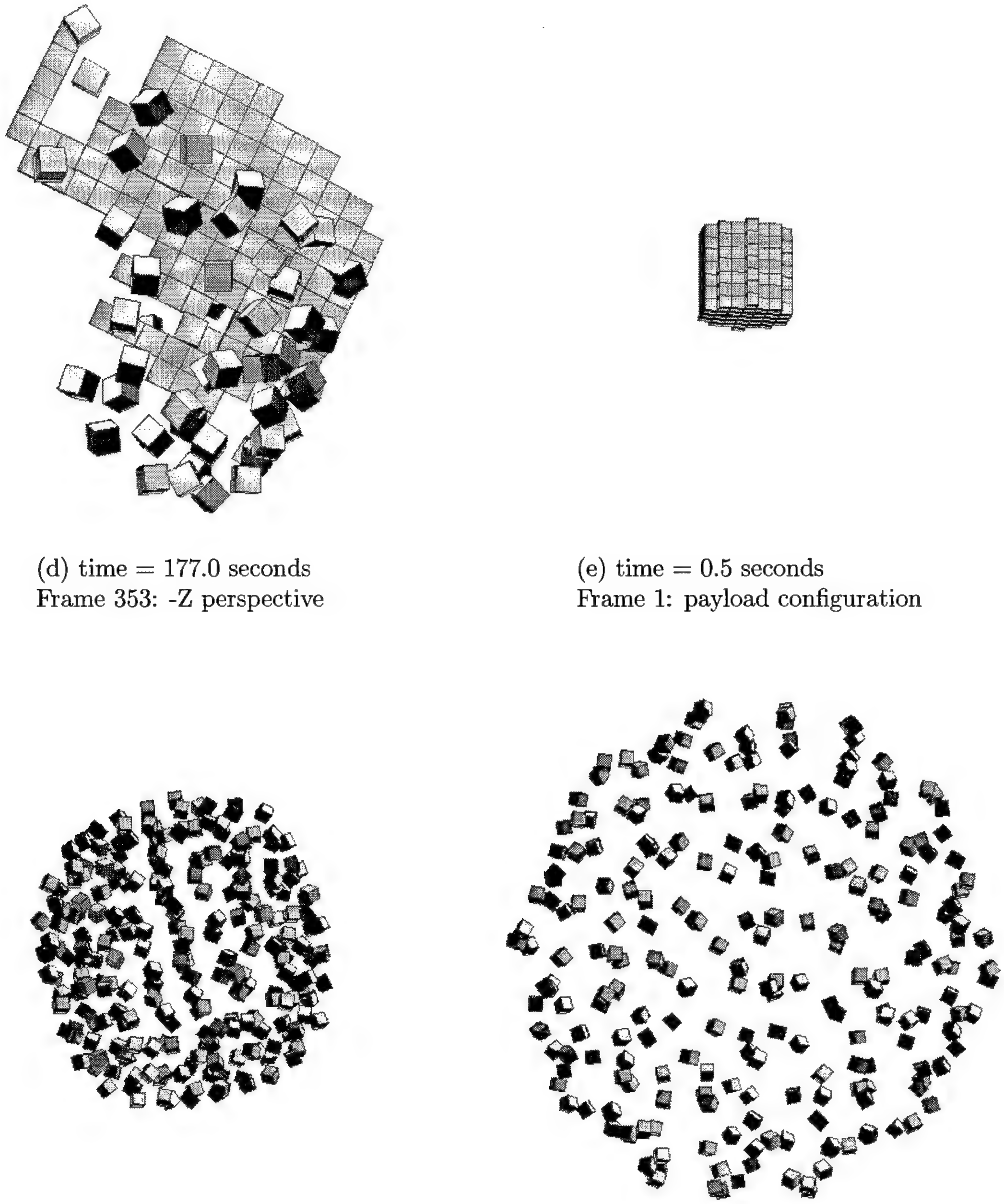
(b) *Paraboloid* [right]
Frame 353

(c) *Paraboloid* [front]
Frame 353

Figure (c) best illustrates paraboloid construction. Notice that the left side on the front view is at a more advanced stage of construction than the left.

Figure 5.15 [a,b,c] inverted paraboloid constructed with low system noise..

Case Study C*: [page 2] 354 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0025$



(d) time = 177.0 seconds
Frame 353: -Z perspective

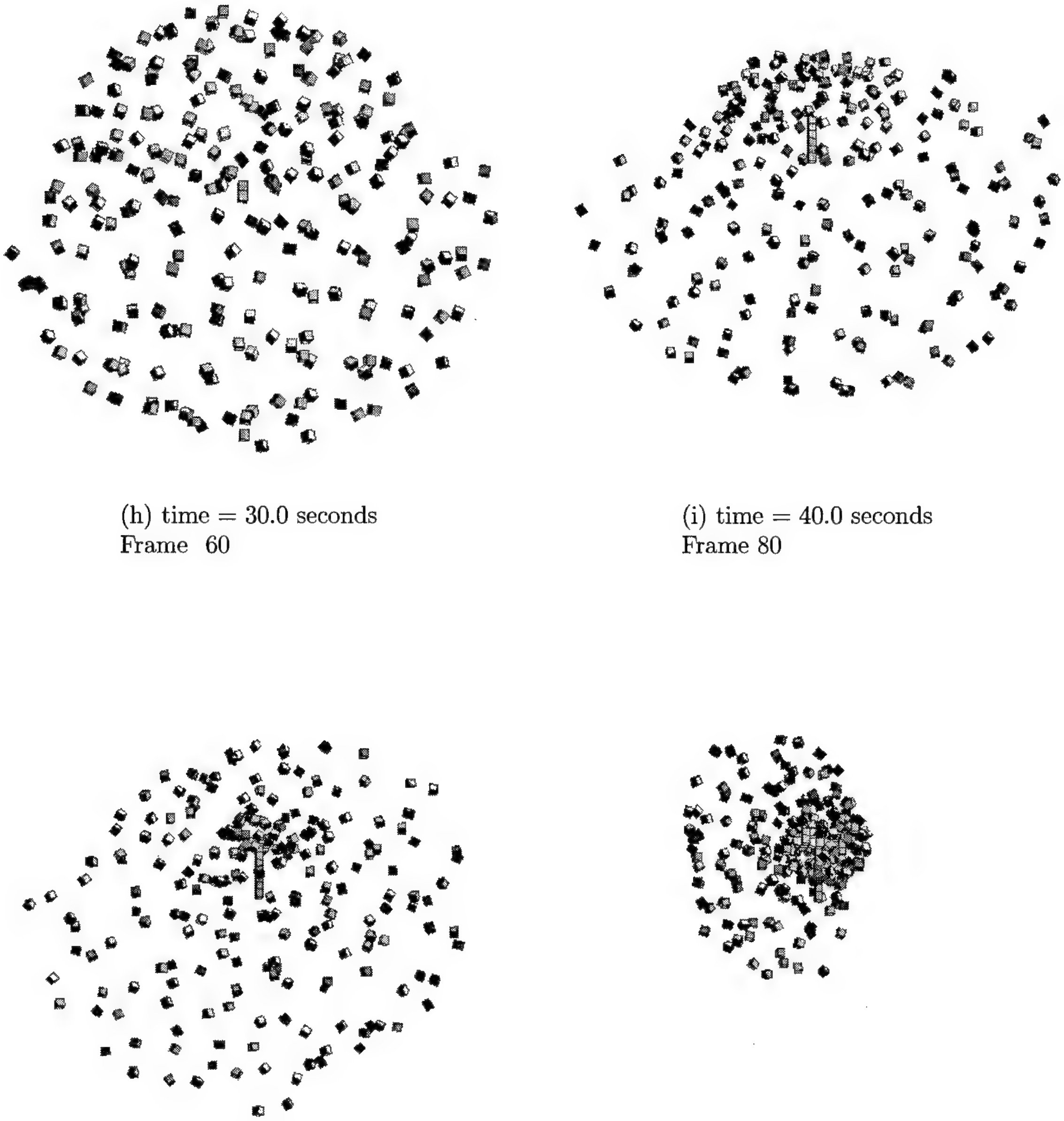
(e) time = 0.5 seconds
Frame 1: payload configuration

(f) time = 10.0 seconds
Frame 20: 40% of Δ_{12} complete

(g) time = 20 seconds
Frame 40: 80% of Δ_{12} complete

Figure 5.16 [d,e,f,g]

Case Study C*:* [page 3] *354 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0025$



(h) time = 30.0 seconds
Frame 60

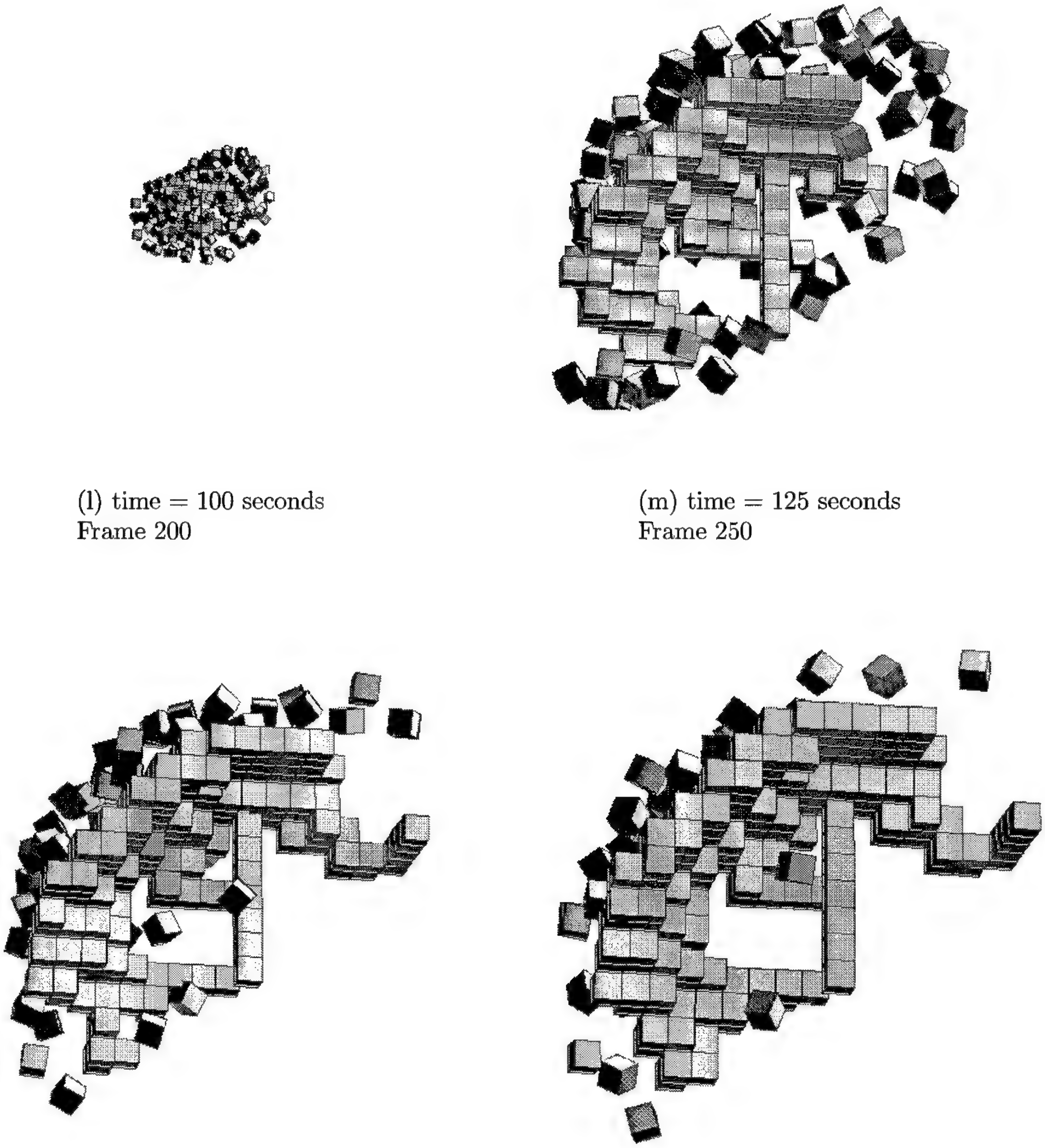
(i) time = 40.0 seconds
Frame 80

(j) time = 50.0 seconds
Frame 100

(k) time = 75.0 seconds
Frame 150

Figure 5.17 [h,i,j,k]

Case Study C*: [page 4] 354 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0025$



(l) time = 100 seconds
Frame 200

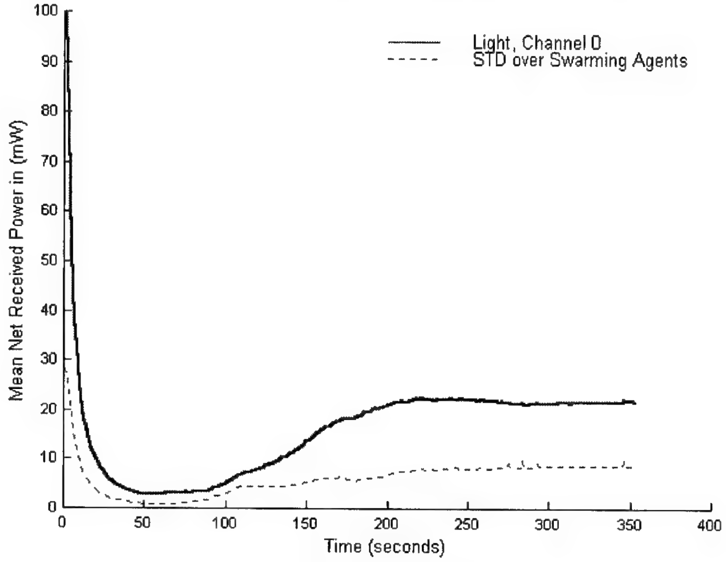
(m) time = 125 seconds
Frame 250

(n) time = 150 seconds
Frame 300

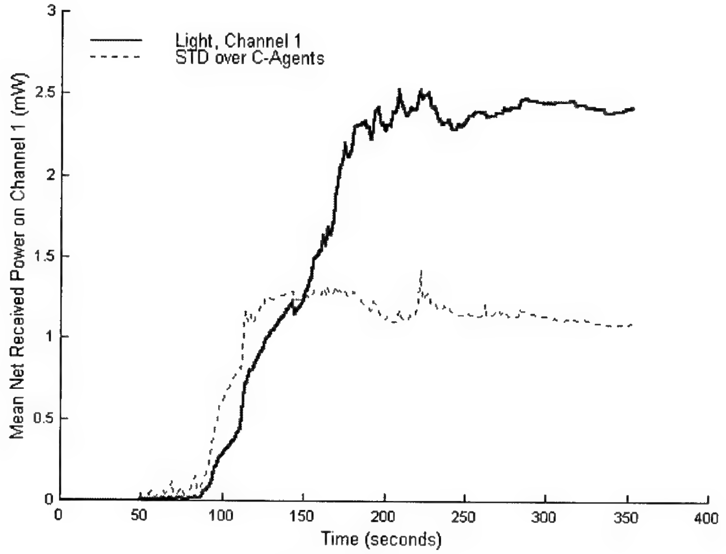
(o) time = 177.0 seconds
Frame 354

Figure 5.18 [l,m,n,o]

Case Study C: *Plots five and six of eight, Tranceiver CH-0 and CH-1.*



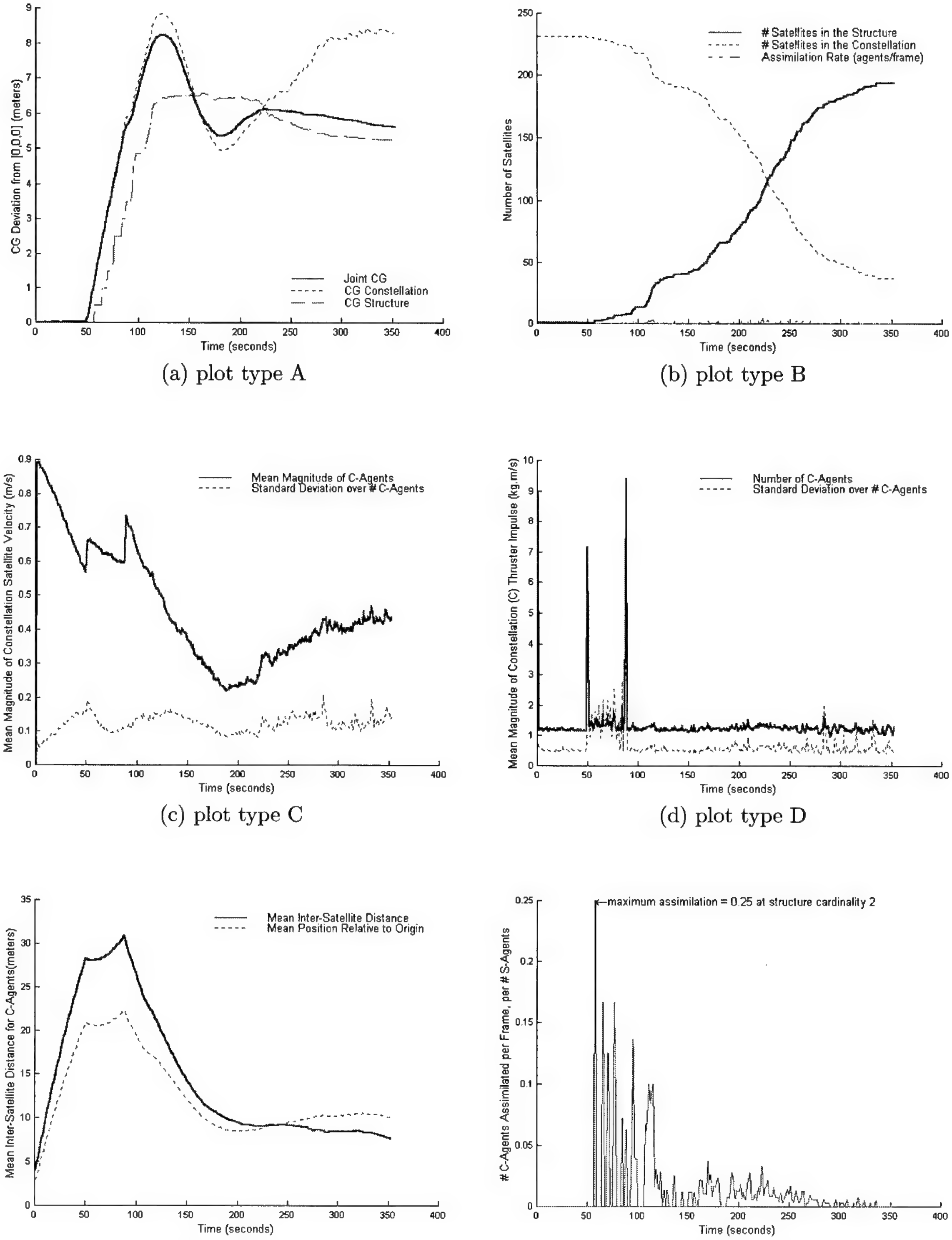
(e) plot type E



(f) plot type F

Figure 5.19 [e,f] Channel 0 and Channel 1 received power as a function of time

Battery C: *plots 1,2,3,4,7, and 8 of 8* *System Noise* $\sigma^2_{xyz} = 0.0025$



(a) plot type A

(b) plot type B

(c) plot type C

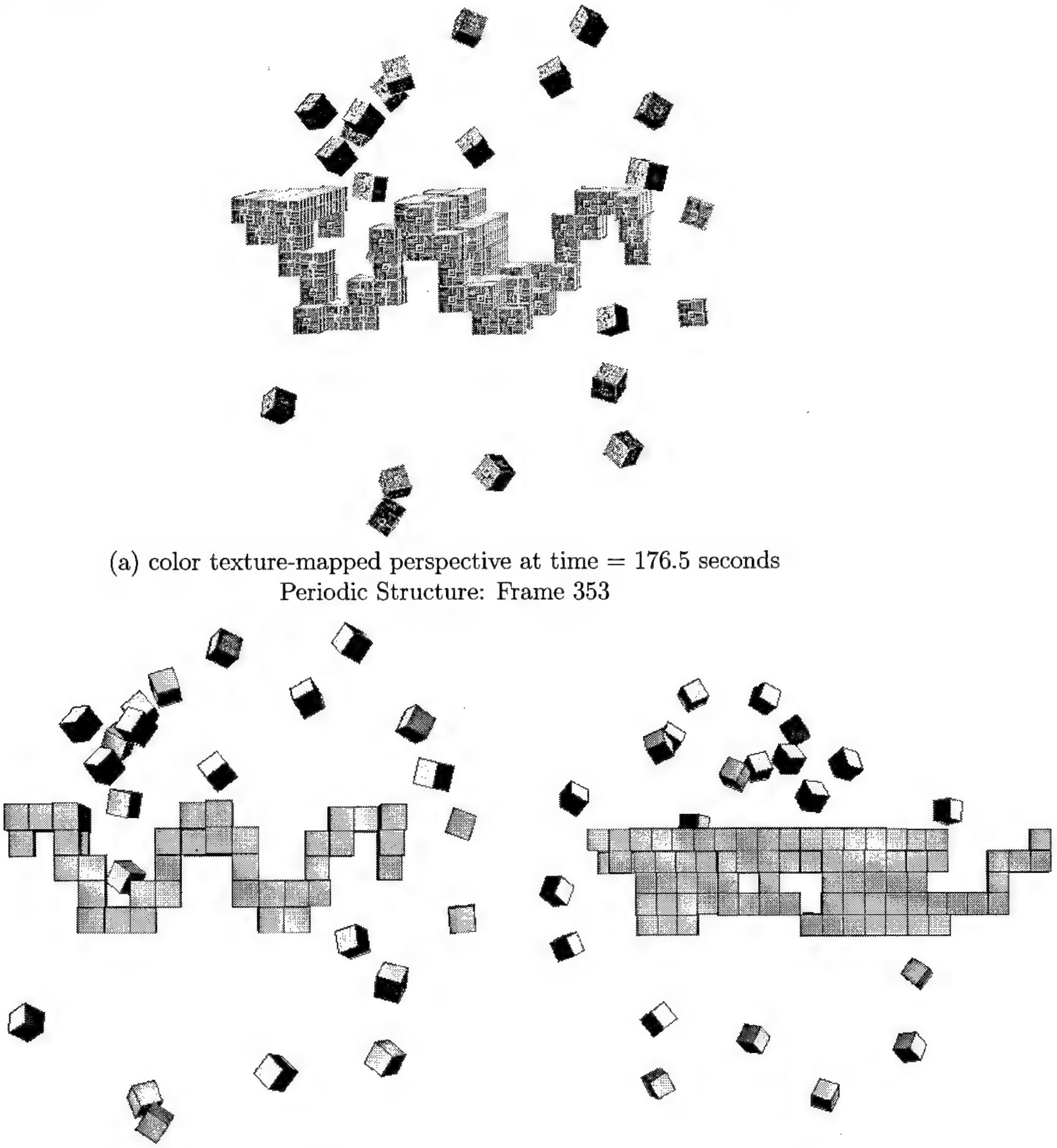
(d) plot type D

(g) plot type G

(h) plot type H

Figure 5.20 Battery C plots [a,b,c,d,g,h]

Case Study D: *355 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0025$



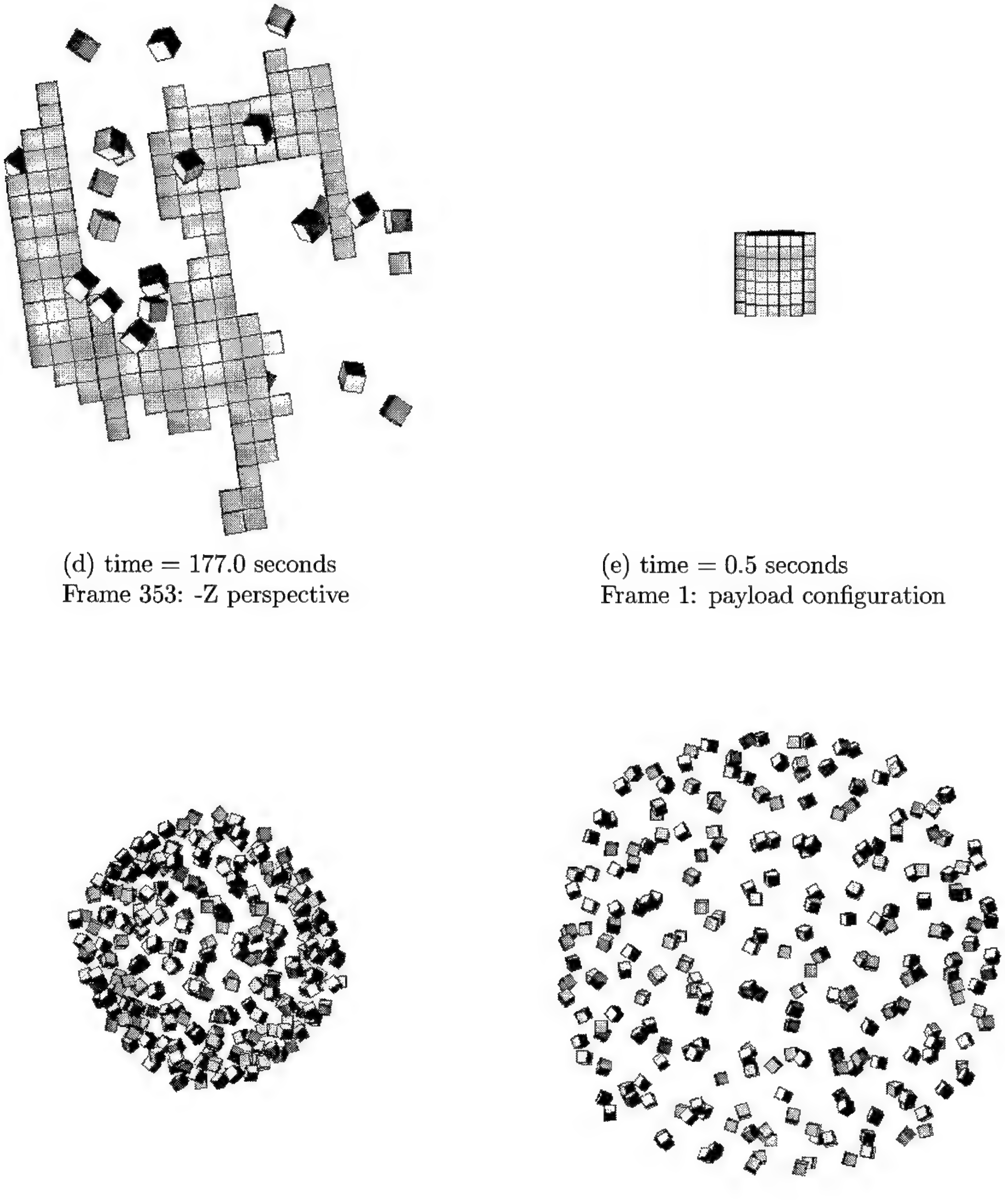
(a) color texture-mapped perspective at time = 176.5 seconds
Periodic Structure: Frame 353

(b) *Paraboloid* [right]
Frame 353

(c) *Paraboloid* [front]
Frame 353

Figure 5.21 [a,b,c] with low system noise at the agent level

Case Study D: *[page 2] 355 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0025$



(d) time = 177.0 seconds
Frame 353: -Z perspective

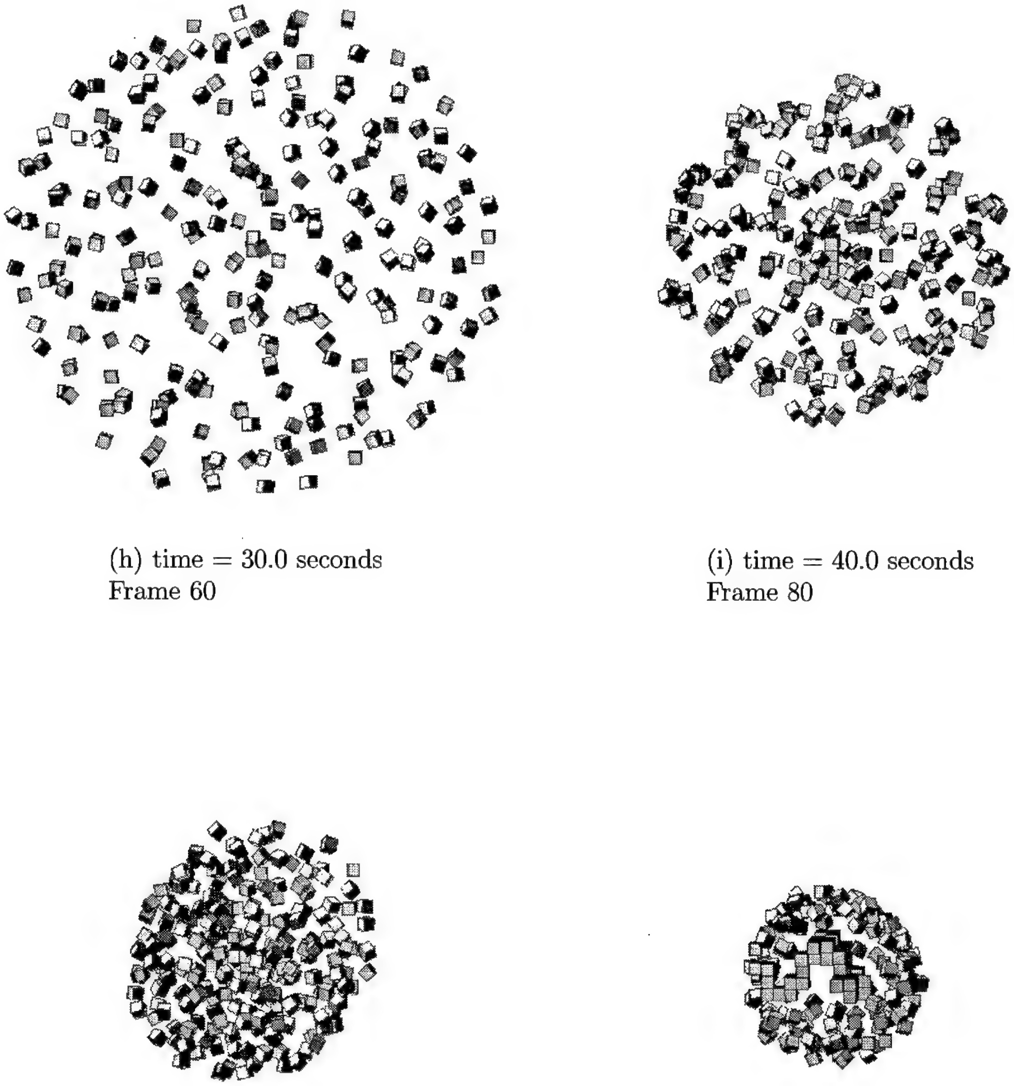
(e) time = 0.5 seconds
Frame 1: payload configuration

(f) time = 10.0 seconds
Frame 20: 40% of Δ_{12} complete

(g) time = 20 seconds
Frame 40: 80% of Δ_{12} complete

Figure 5.22 [d,e,f,g]

Case Study D: *[page 3] 355 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0025$



(h) time = 30.0 seconds
Frame 60

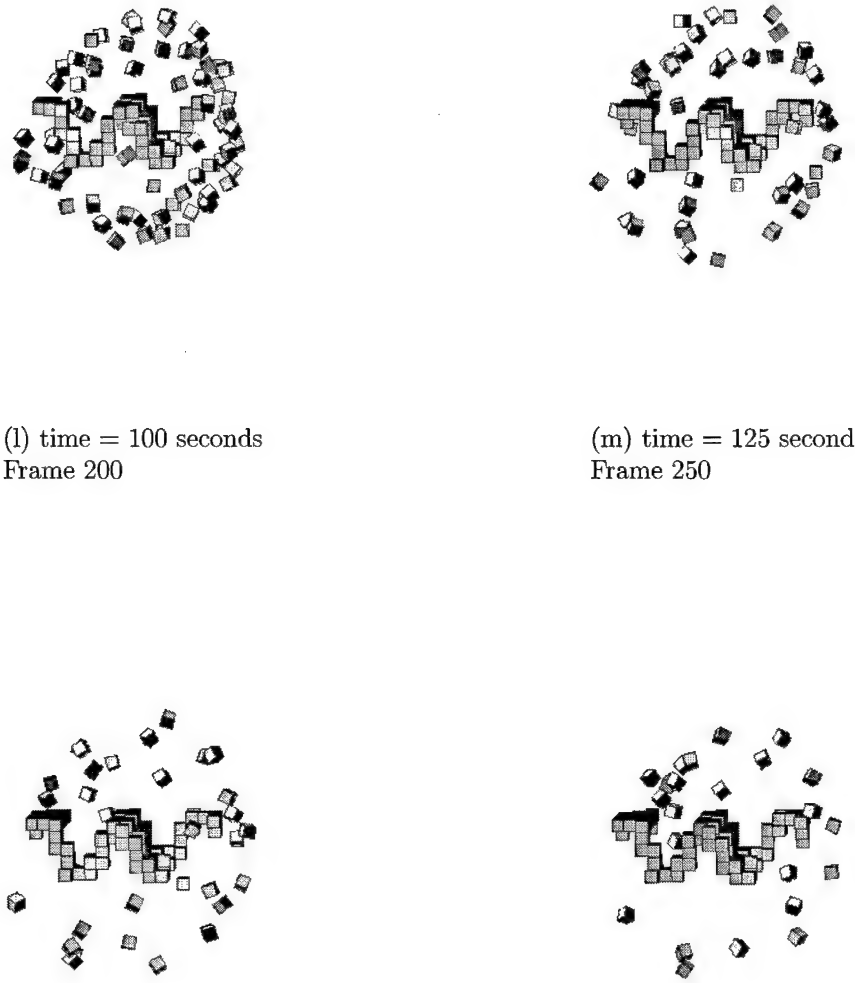
(i) time = 40.0 seconds
Frame 80

(j) time = 50.0 seconds
Frame 100

(k) time = 75.0 seconds
Frame 150

Figure 5.23 [h,i,j,k]

Case Study D: *[page 4] 355 frames, 231 satellite constellation with system noise* $\sigma_{xyz}^2 = 0.0025$



(l) time = 100 seconds
Frame 200

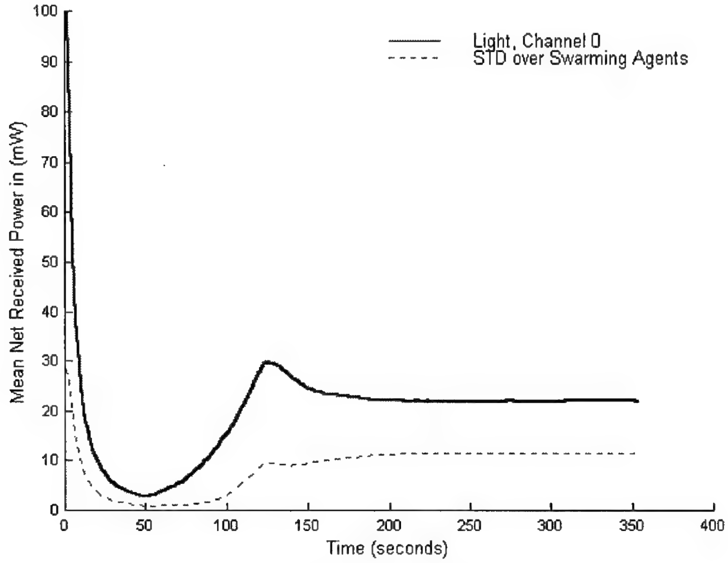
(m) time = 125 seconds
Frame 250

(n) time = 150 seconds
Frame 300

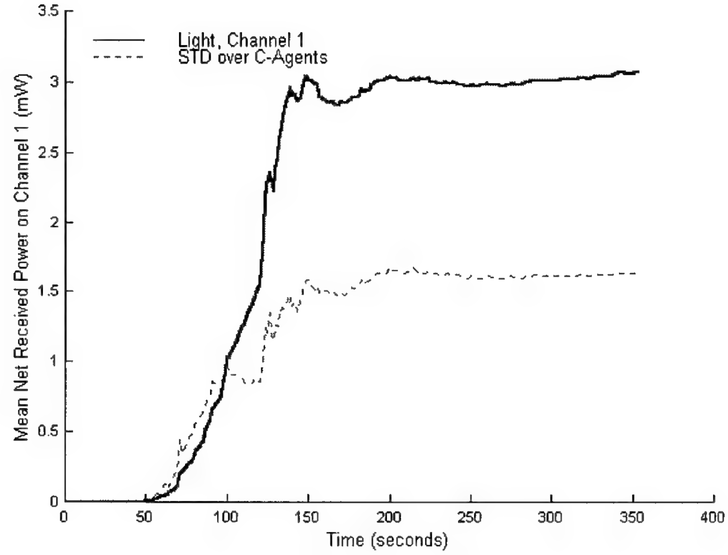
(o) time = 177.0 seconds
Frame 354

Figure 5.24 [l,m,n,o]

Case Study D*: Plots five and six of eight, Tranceiver CH-0 and CH-1.*



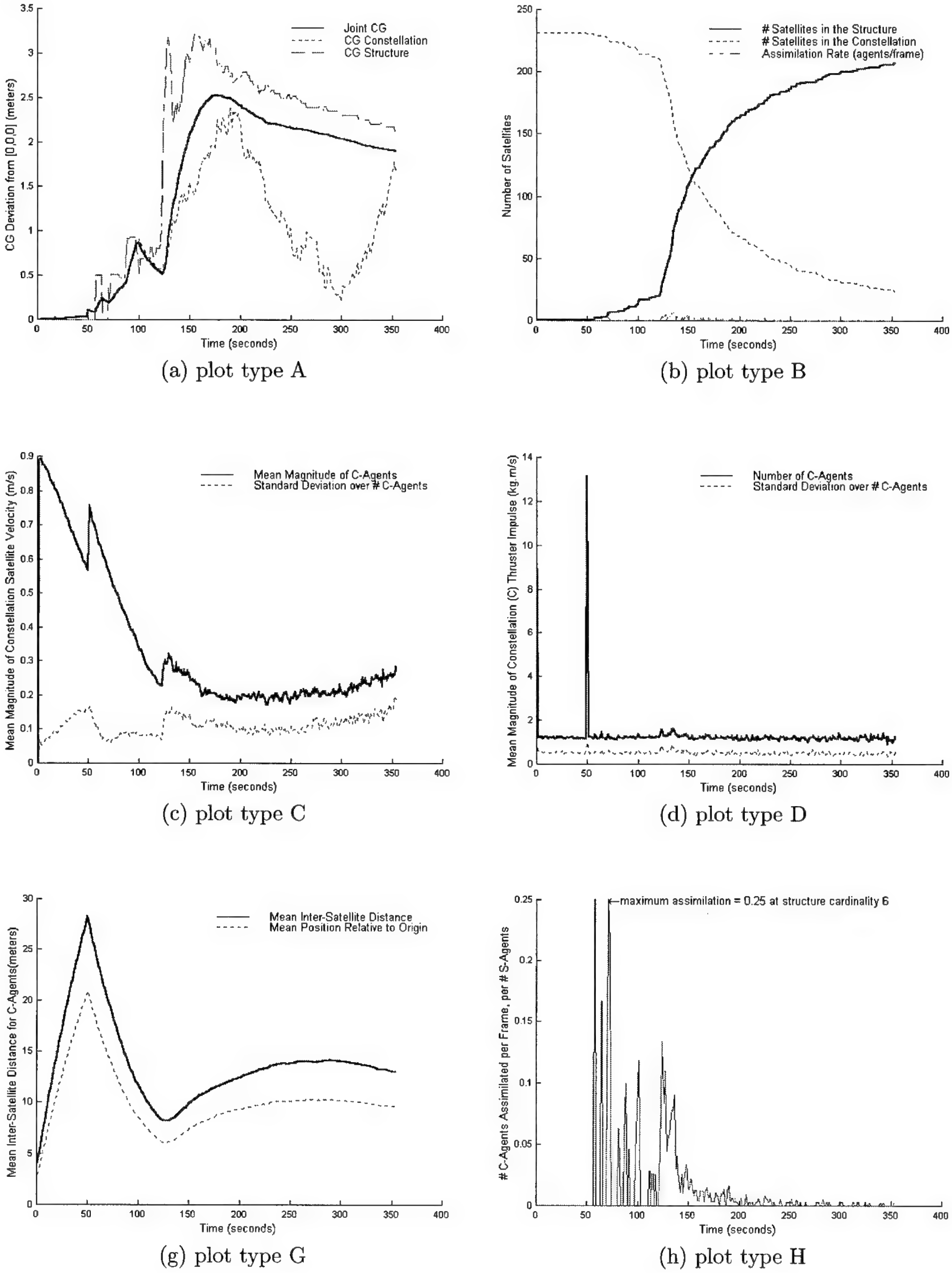
(e) plot type E



(f) plot type F

Figure 5.25 [e,f] Channel 0 and Channel 1 received power as a function of time

Battery D: *plots 1,2,3,4,7, and 8 of 8* *System Noise* $\sigma^2_{xyz} = 0.0025$



(a) plot type A

(b) plot type B

(c) plot type C

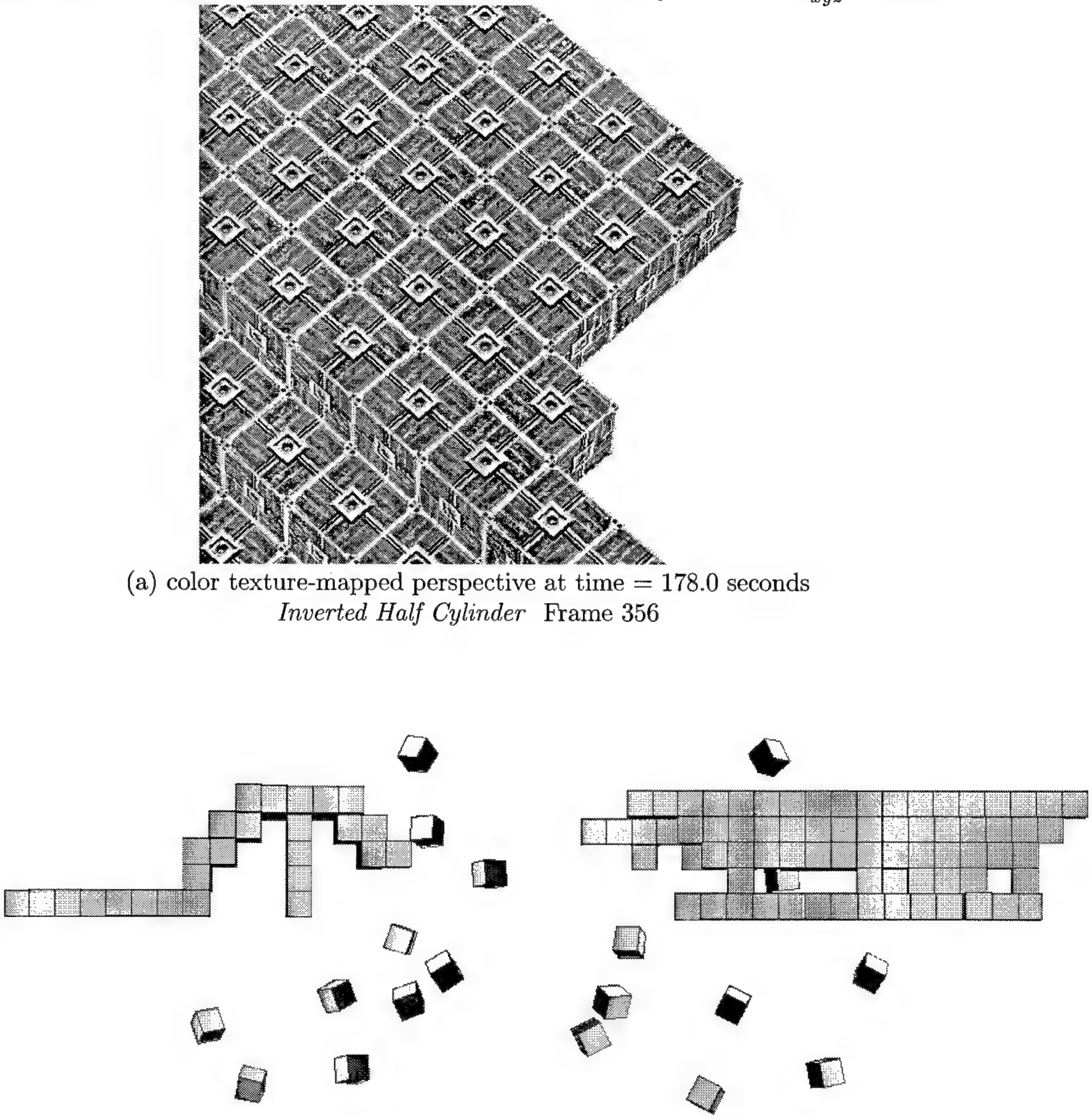
(d) plot type D

(g) plot type G

(h) plot type H

Figure 5.26 Battery D plots [a,b,c,d,g,h]

Case Study E: *356 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0040$



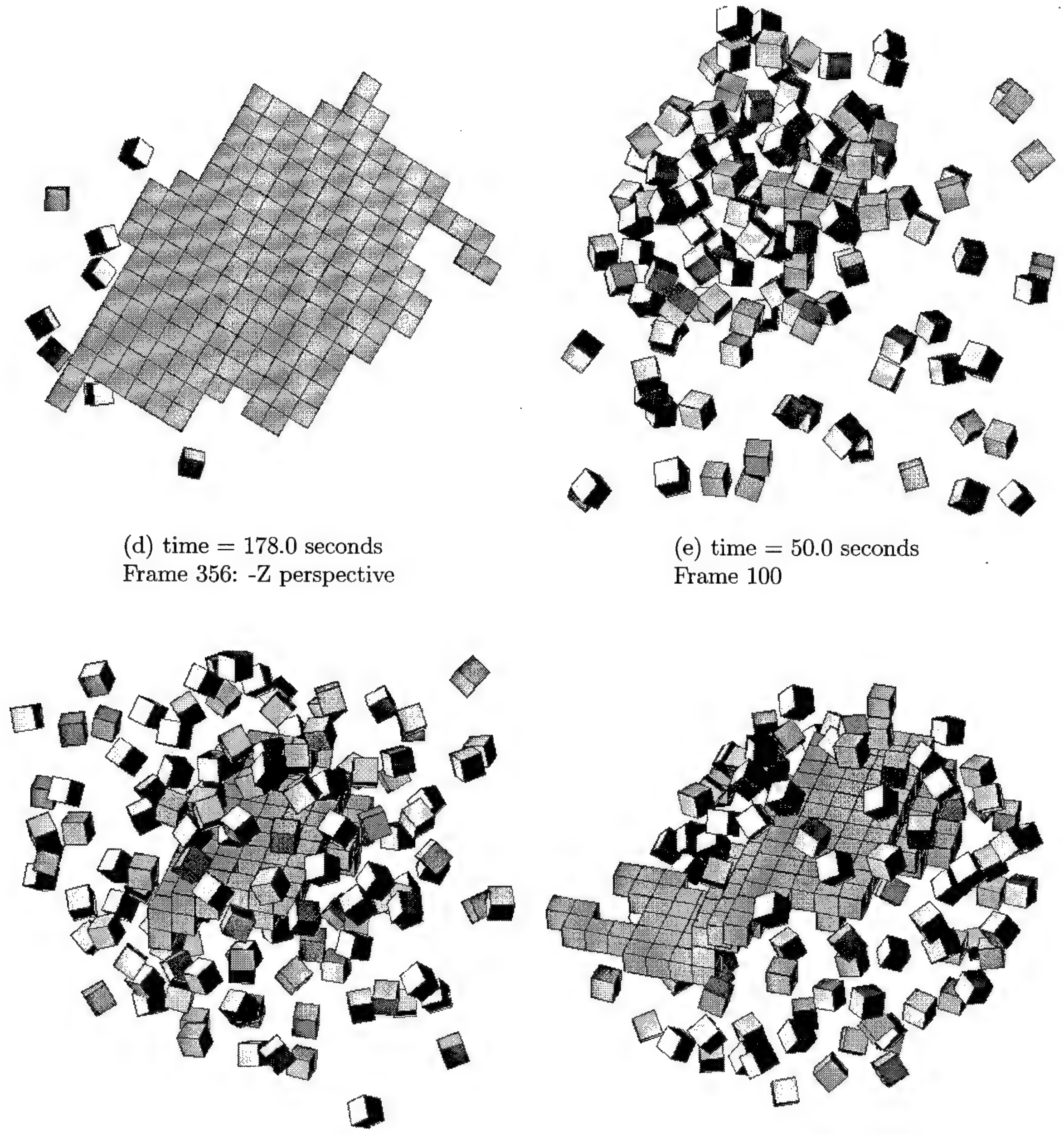
(a) color texture-mapped perspective at time = 178.0 seconds
Inverted Half Cylinder Frame 356

(b) *Paraboloid* [right view]
Frame 356

(c) *Paraboloid* [front view]
Frame 356

Figure 5.27 [a,b,c] with low system noise at the agent level

Case Study E*:* [page 2] *356 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0040$



(d) time = 178.0 seconds
Frame 356: -Z perspective

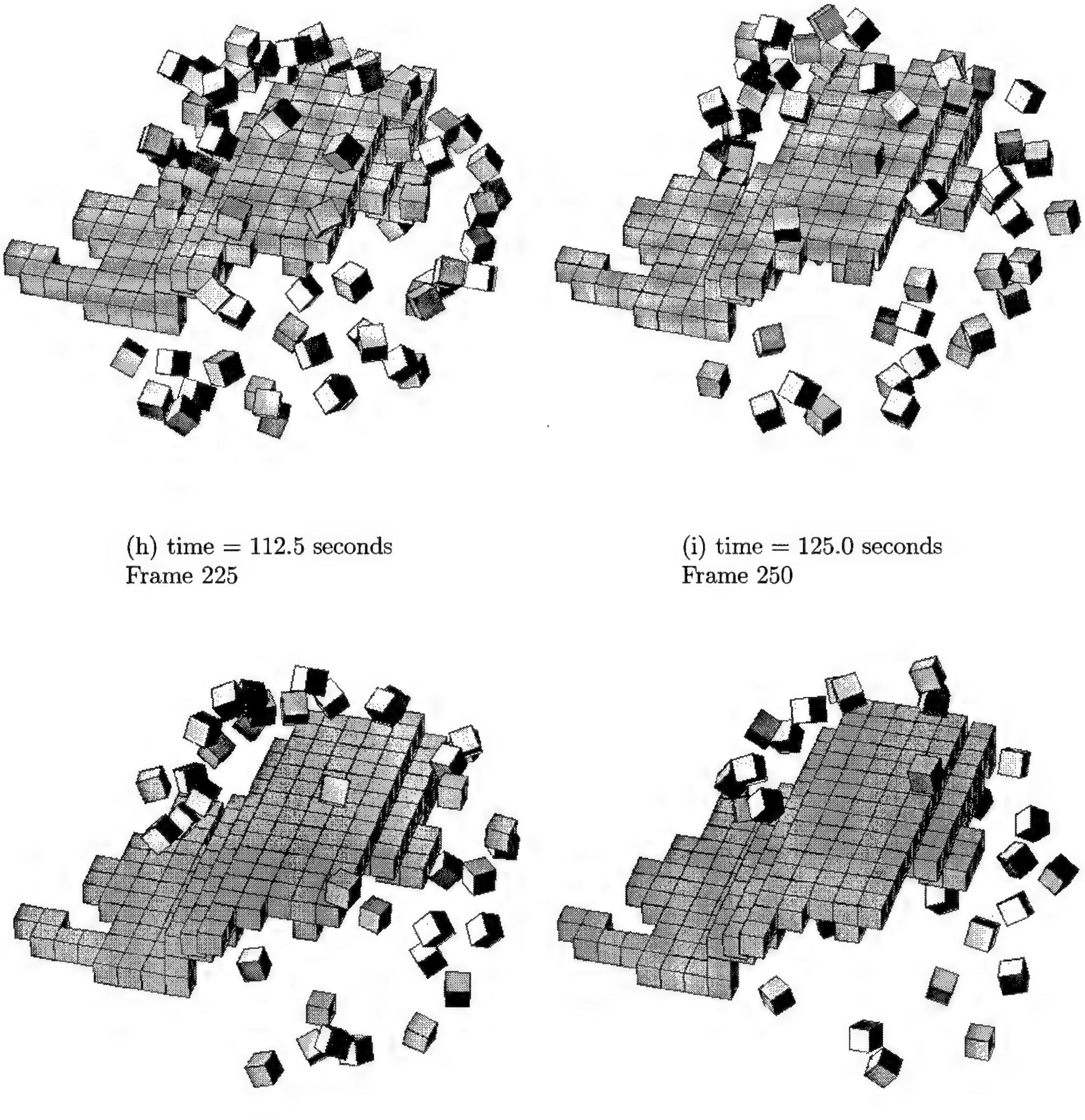
(e) time = 50.0 seconds
Frame 100

(f) time = 75.0 seconds
Frame 150

(g) time = 100 seconds
Frame 200

Figure 5.28 [d,e,f,g]

Case Study E*:* [page 3] *356 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0040$



(h) time = 112.5 seconds
Frame 225

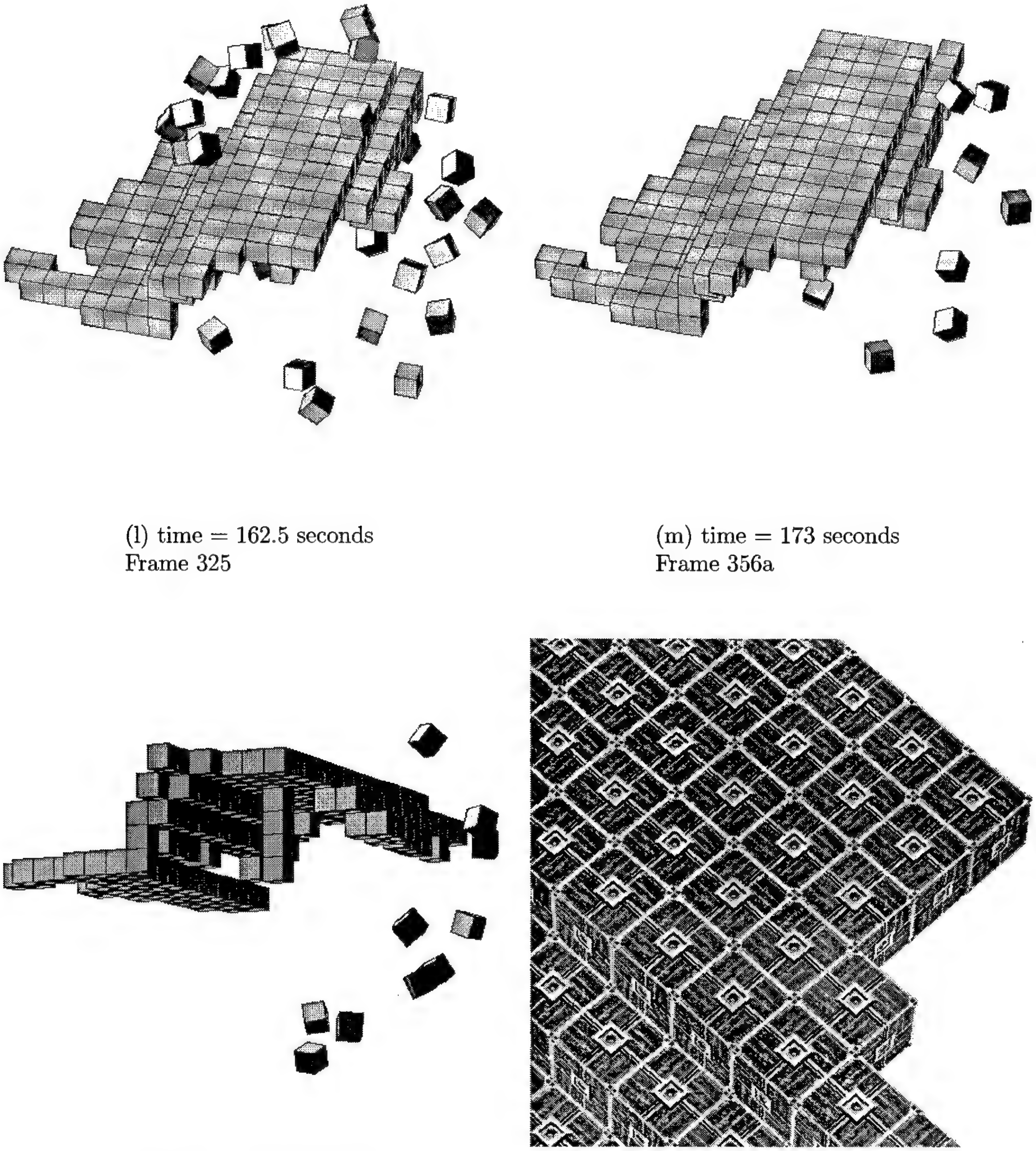
(i) time = 125.0 seconds
Frame 250

(j) time = 137.5 seconds
Frame 275

(k) time = 150.0 seconds
Frame 300

Figure 5.29 [h,i,j,k]

Case Study E*:* [page 4] *356 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0040$



(l) time = 162.5 seconds
Frame 325

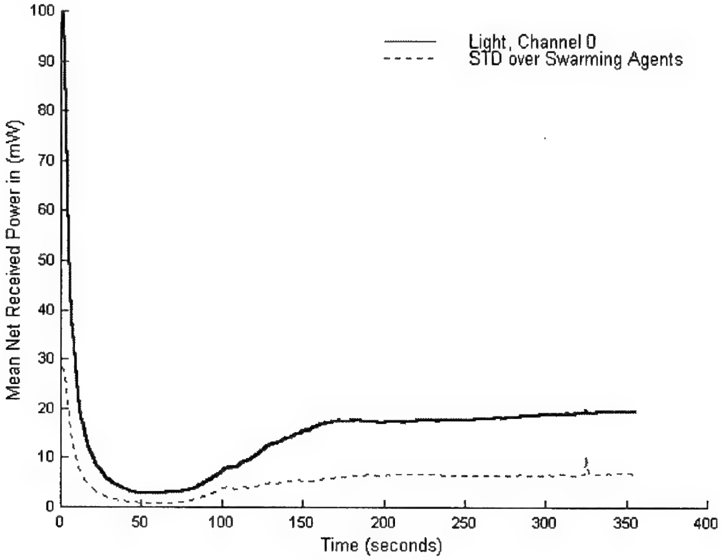
(m) time = 173 seconds
Frame 356a

(n) time = 173 seconds
Frame 356b

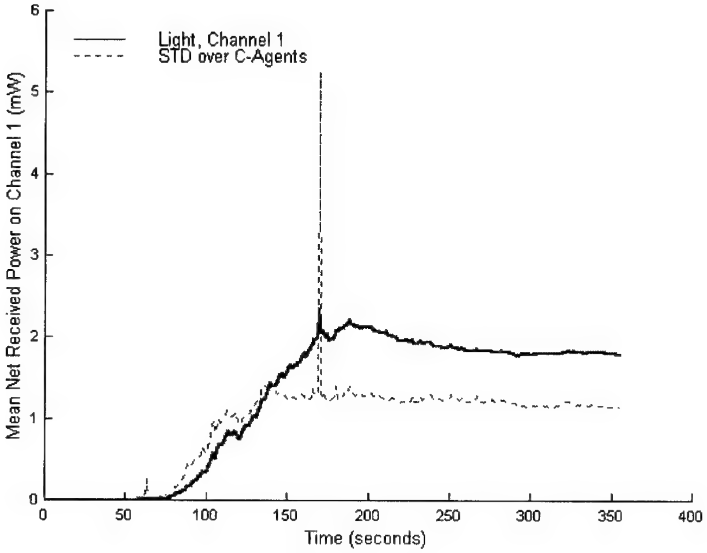
(o) time = 173.0 seconds
Frame 356

Figure 5.30 [l,m,n,o]

Case Study E*: Plots five and six of eight, Tranceiver CH-0 and CH-1.*



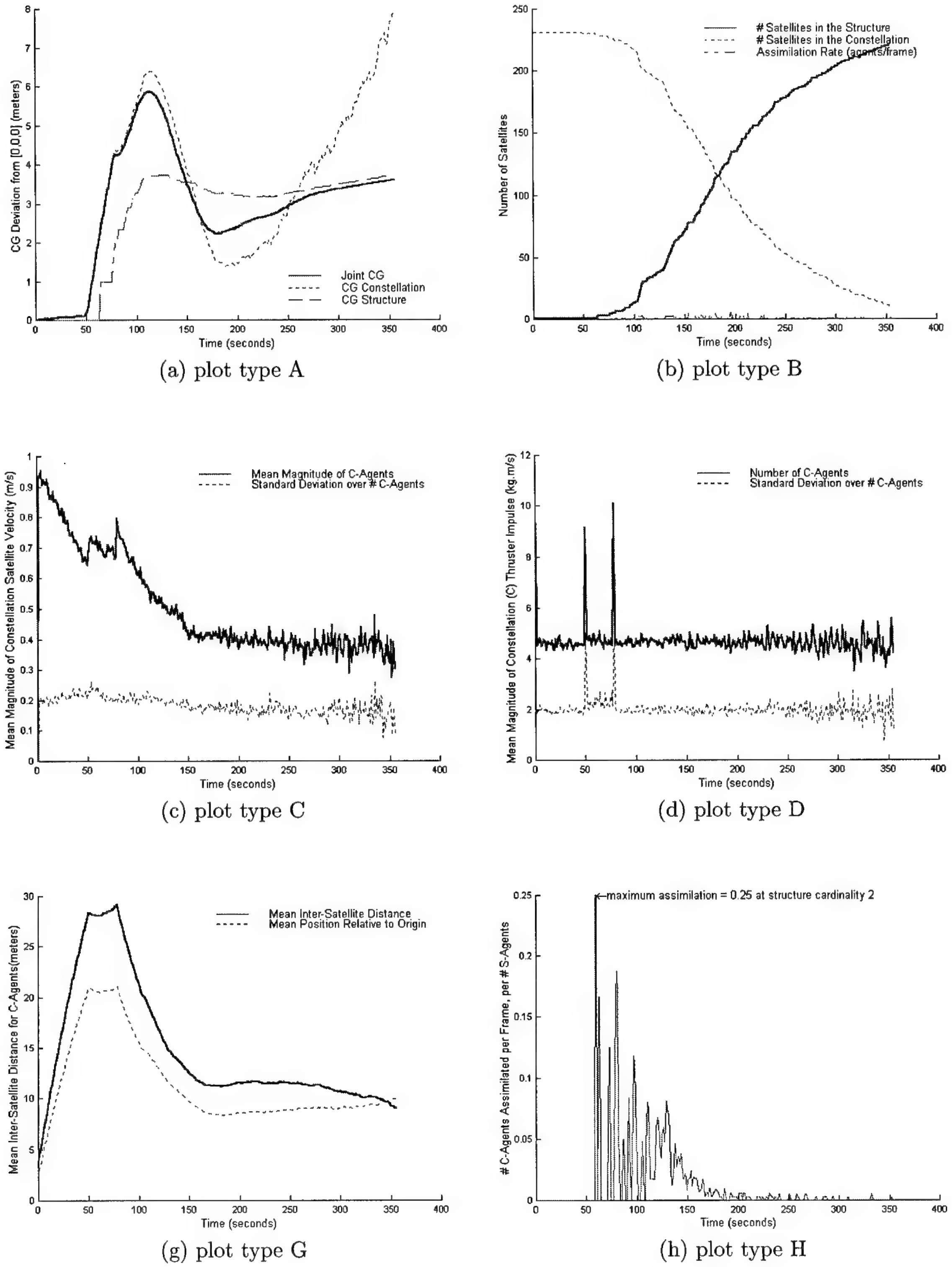
(e) plot type **E**



(f) plot type **F**

Figure 5.31 [e,f] Channel 0 and Channel 1 received power as a function of time

Battery E: *plots 1,2,3,4,7, and 8 of 8* *System Noise* $\sigma_{xyz}^2 = 0.0040$



(a) plot type A

(b) plot type B

(c) plot type C

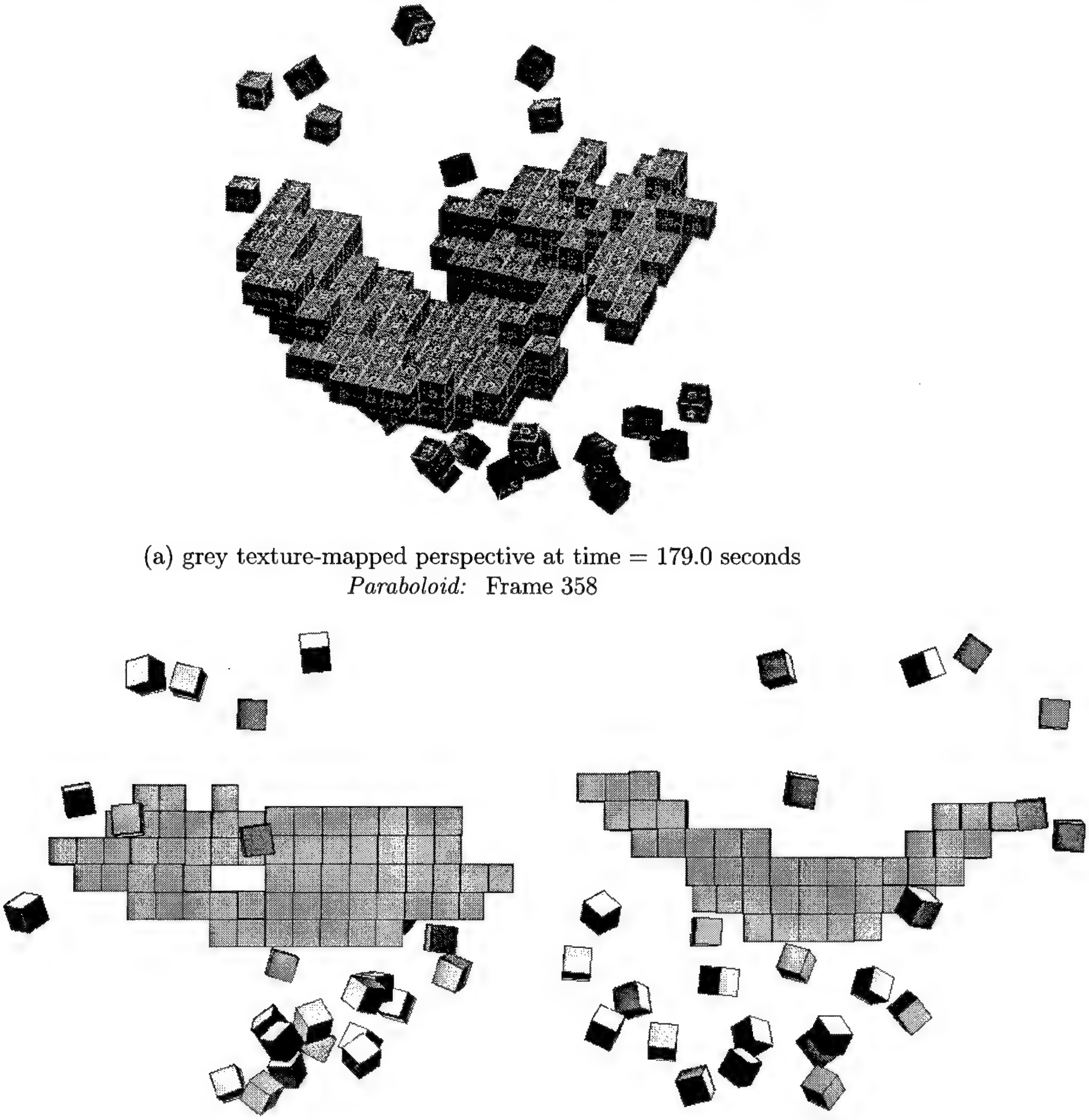
(d) plot type D

(g) plot type G

(h) plot type H

Figure 5.32 Battery E plots [a,b,c,d,g,h]

Case Study F*: 358 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0040$



(a) grey texture-mapped perspective at time = 179.0 seconds

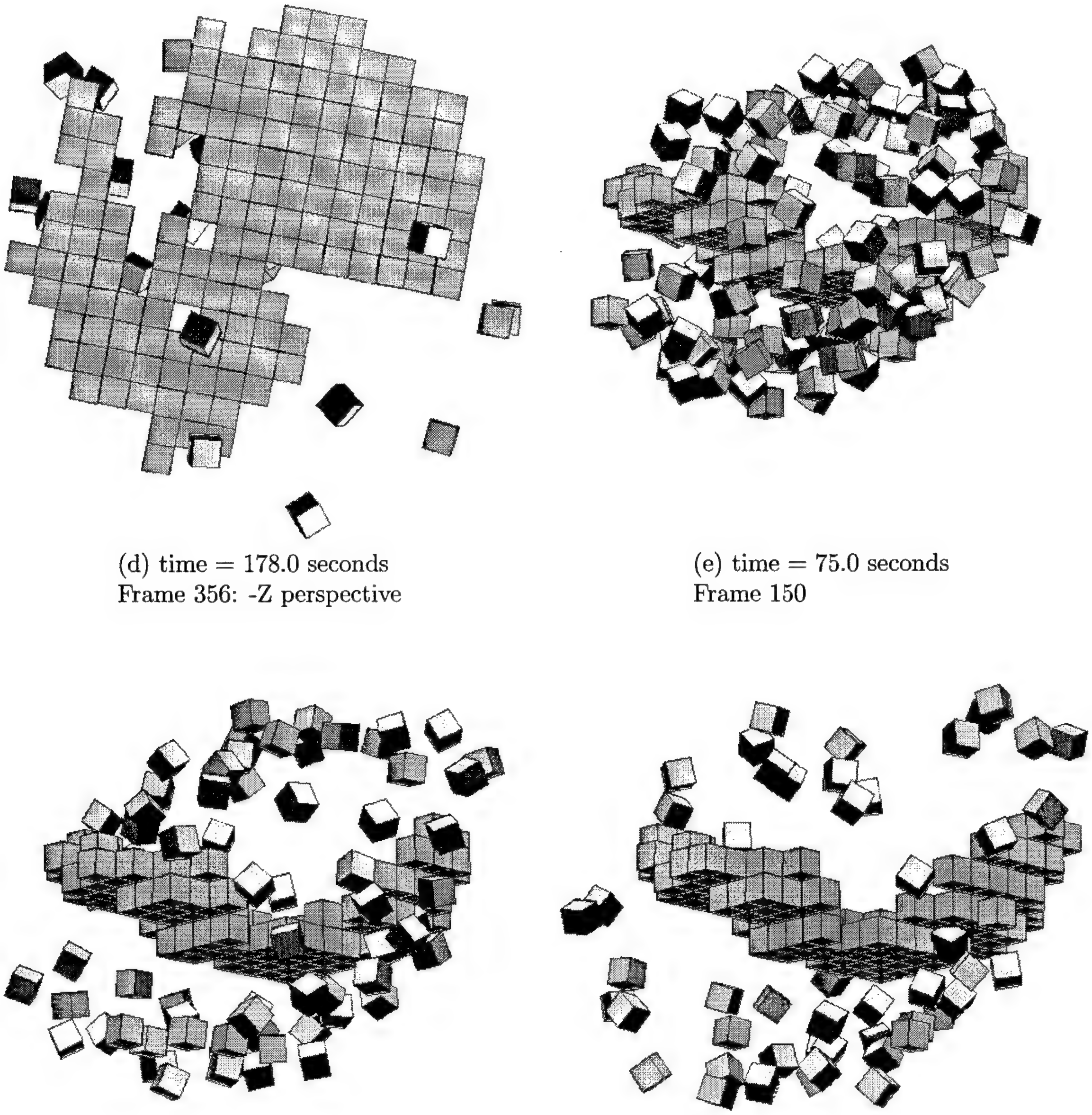
Paraboloid: Frame 358

(b) *Paraboloid* [right view]
Frame 358

(c) *Paraboloid* [front view]
Frame 358

Figure 5.33 [a,b,c] with low system noise at the agent level

Case Study F*:* [page 2] *358 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0040$



(d) time = 178.0 seconds
Frame 356: -Z perspective

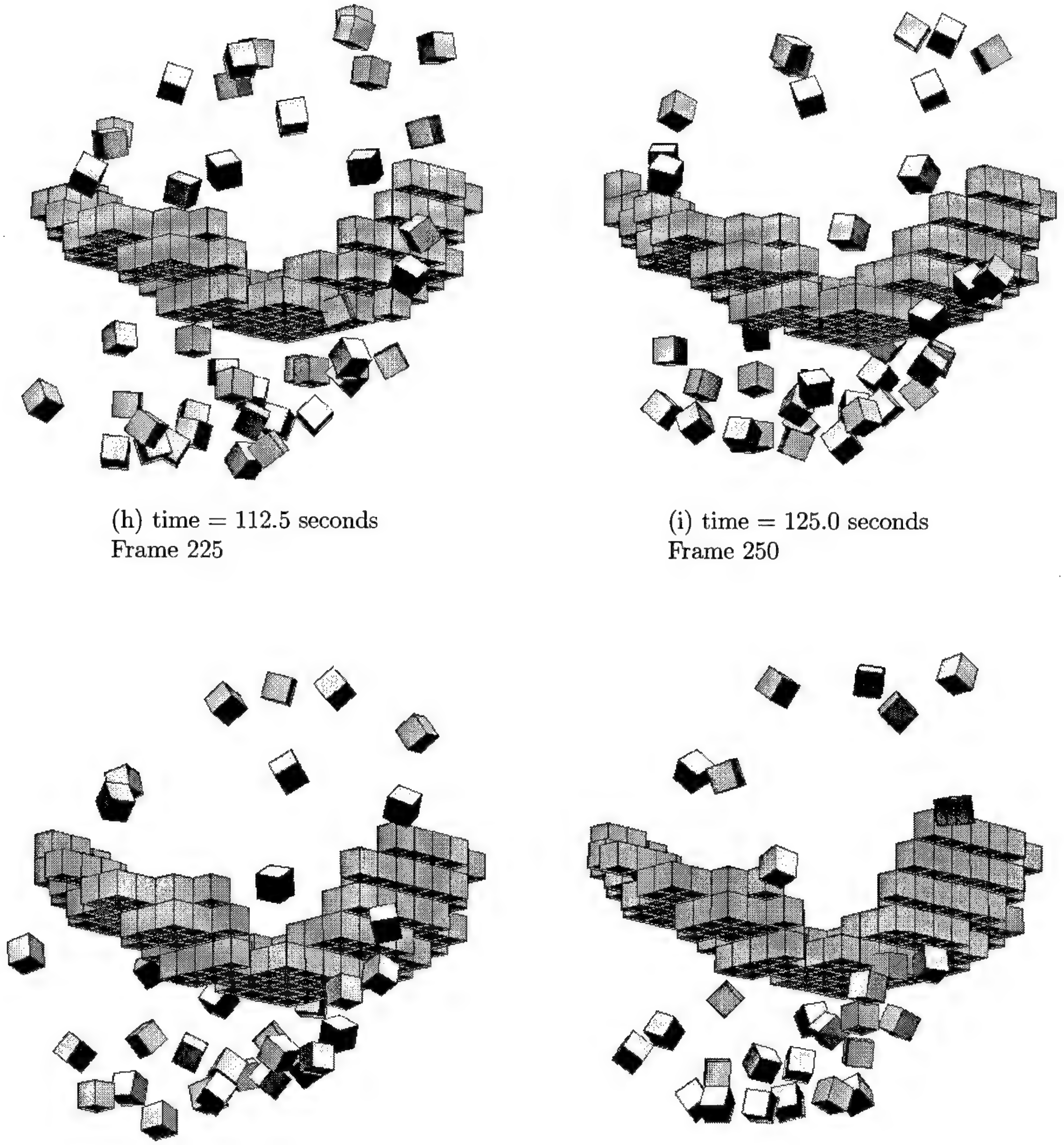
(e) time = 75.0 seconds
Frame 150

(f) time = 87.5 seconds
Frame 175

(g) time = 100.0 seconds
Frame 200

Figure 5.34 [d,e,f,g]

Case Study F*:* [page 3] *358 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0040$



(h) time = 112.5 seconds
Frame 225

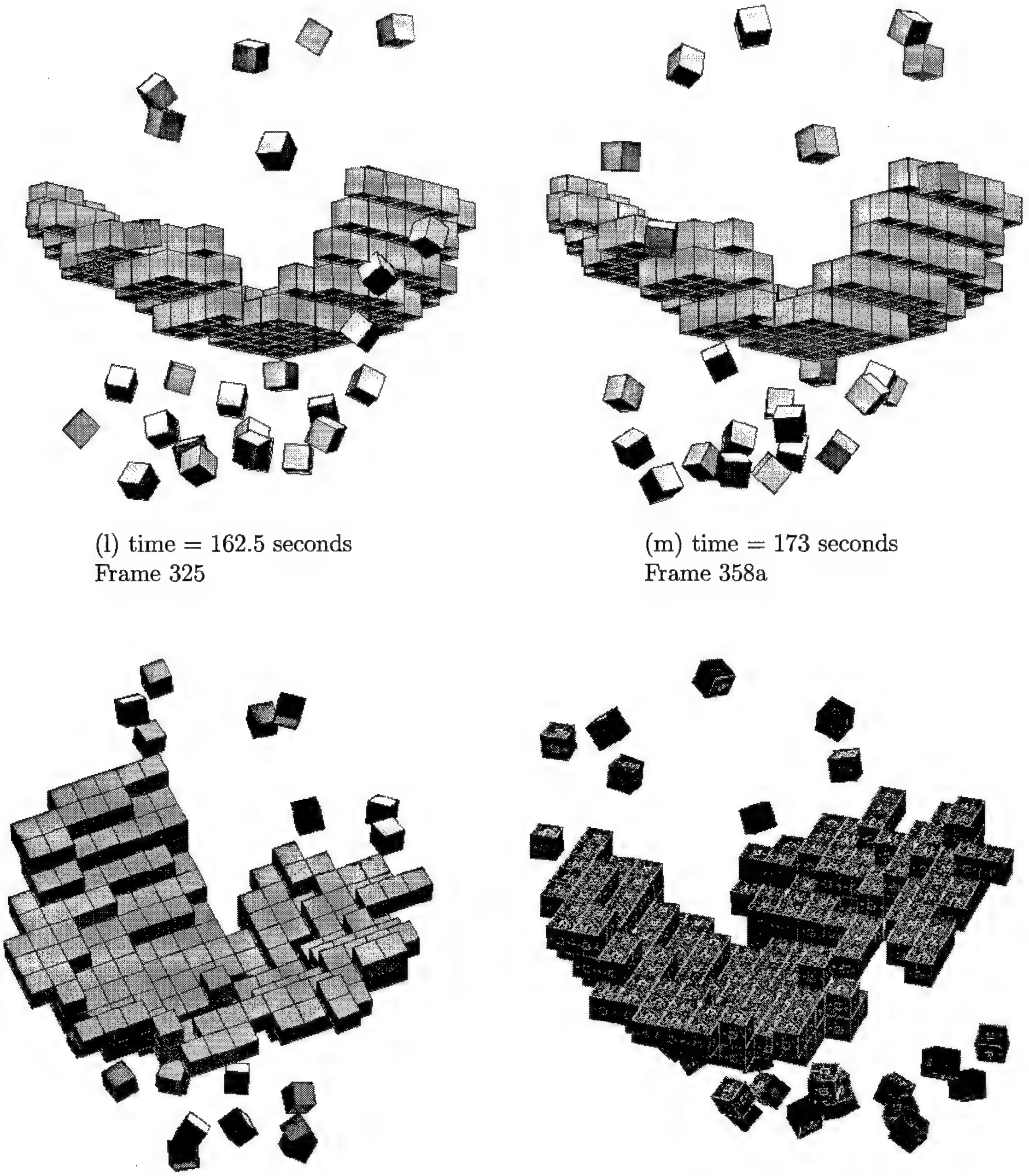
(i) time = 125.0 seconds
Frame 250

(j) time = 137.5 seconds
Frame 275

(k) time = 150.0 seconds
Frame 300

Figure 5.35 [h,i,j,k]

Case Study F: [page 4] *358 frames, 231 satellite constellation with system noise* $\sigma_{xyz}^2 = 0.0040$



(l) time = 162.5 seconds
Frame 325

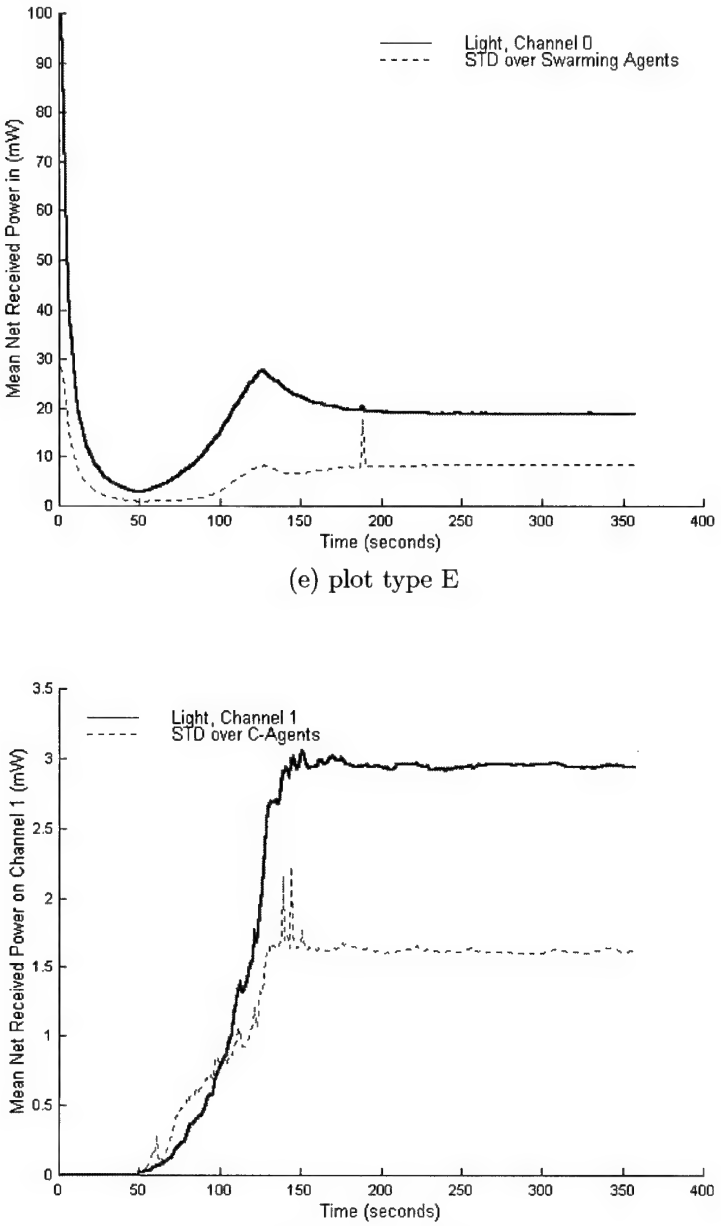
(m) time = 173 seconds
Frame 358a

(n) time = 173 seconds
Frame 358b

(o) time = 173.0 seconds
Frame 358

Figure 5.36 [l,m,n,o]

Case Study F: *Plots five and six of eight, Tranceiver CH-0 and CH-1.*

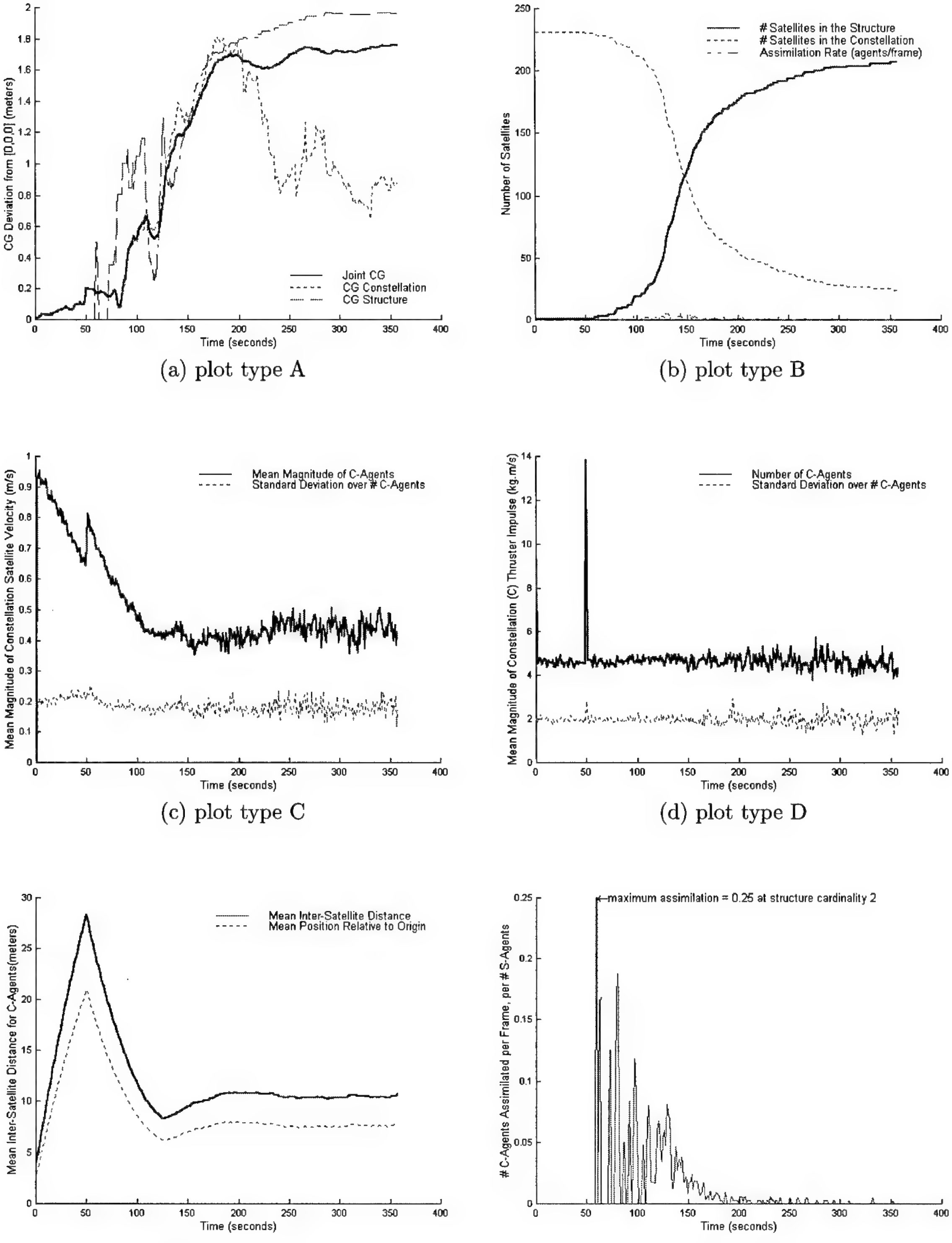


(e) plot type E

(f) plot type F

Figure 5.37 [e,f] Channel 0 and Channel 1 received power as a function of time

Battery F: *plots 1,2,3,4,7, and 8 of 8* *System Noise* $\sigma^2_{xyz} = 0.0040$



(a) plot type A

(b) plot type B

(c) plot type C

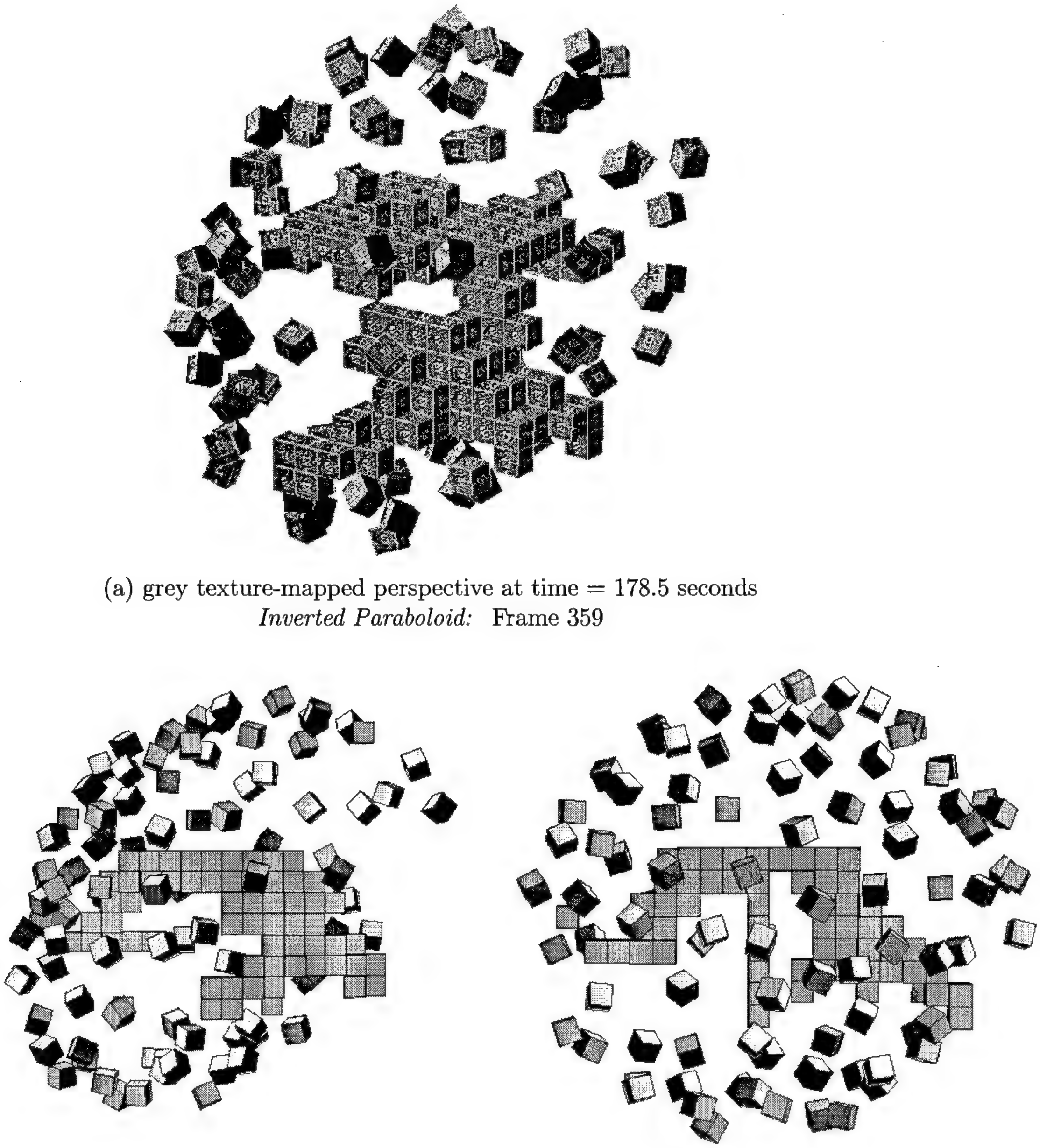
(d) plot type D

(g) plot type G

(h) plot type H

Figure 5.38 Battery F plots [a,b,c,d,g,h]

Case Study G*: 359 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0040$



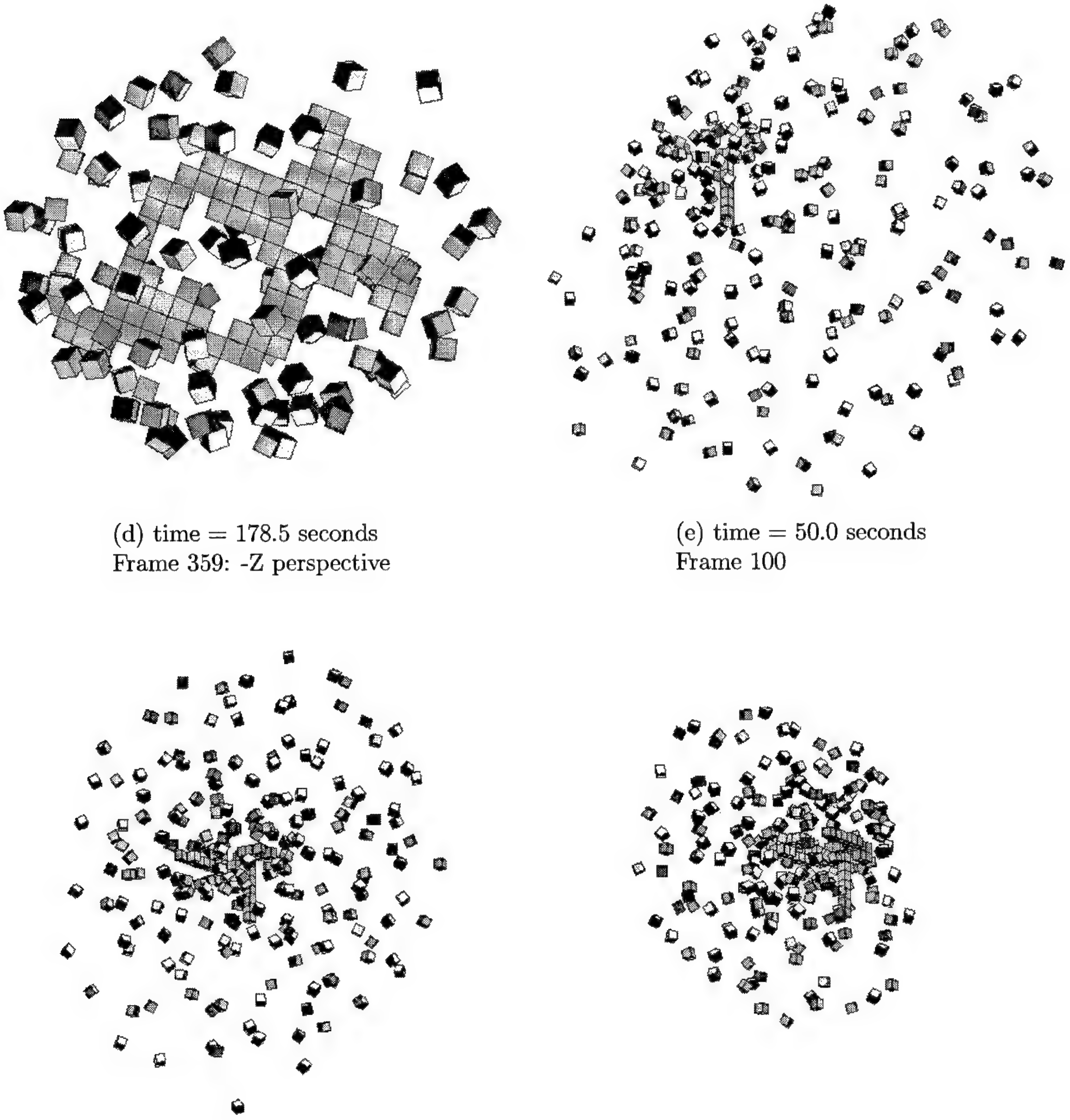
(a) grey texture-mapped perspective at time = 178.5 seconds
Inverted Paraboloid: Frame 359

(b) *Inverted Paraboloid* [right view]
Frame 359

(c) *Inverted Paraboloid* [front view]
frame 359

Figure 5.39 [a,b,c] with low system noise at the agent level

Case Study G: [page 2] *359 frames, 231 satellite constellation with system noise* $\sigma_{xyz}^2 = 0.0040$



(d) time = 178.5 seconds
Frame 359: -Z perspective

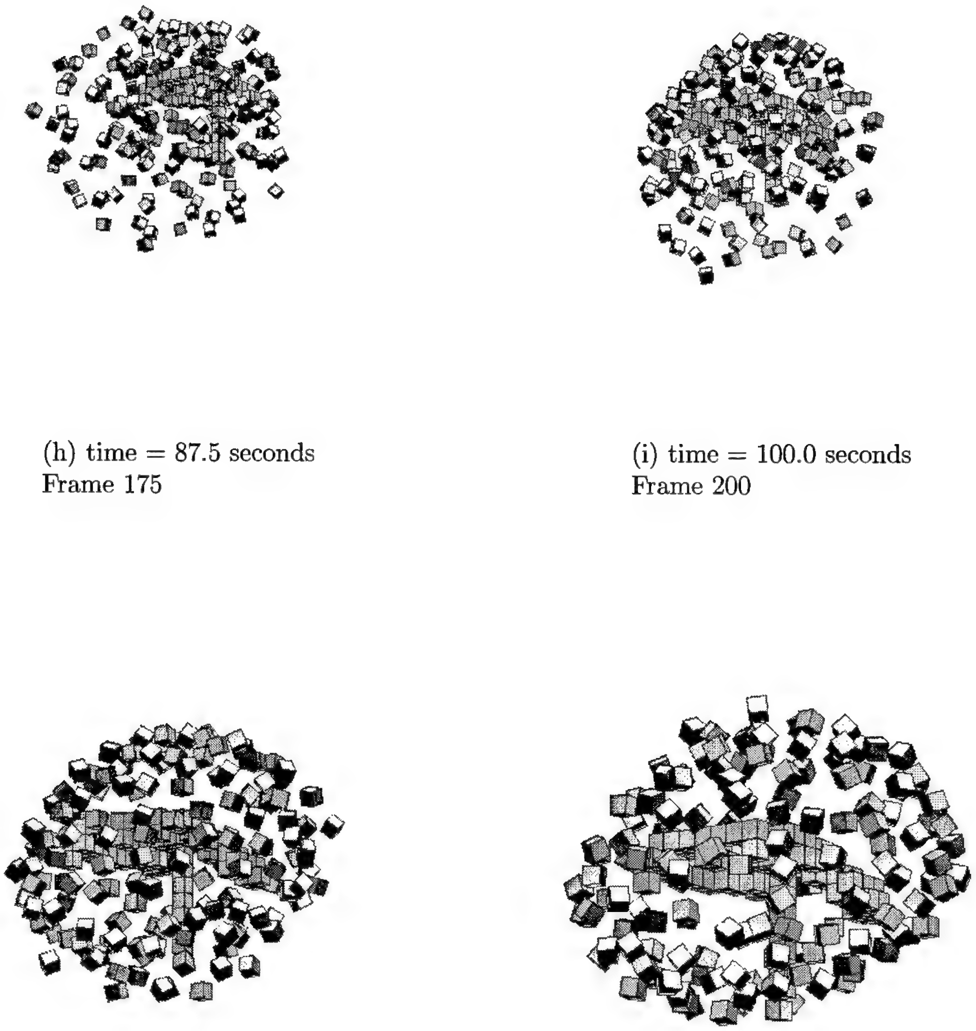
(e) time = 50.0 seconds
Frame 100

(f) time = 113.5 seconds
Frame 125

(g) time = 75.0 seconds
Frame 150

Figure 5.40 [d,e,f,g]

Case Study G: [page 3] *359 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0040$



(h) time = 87.5 seconds
Frame 175

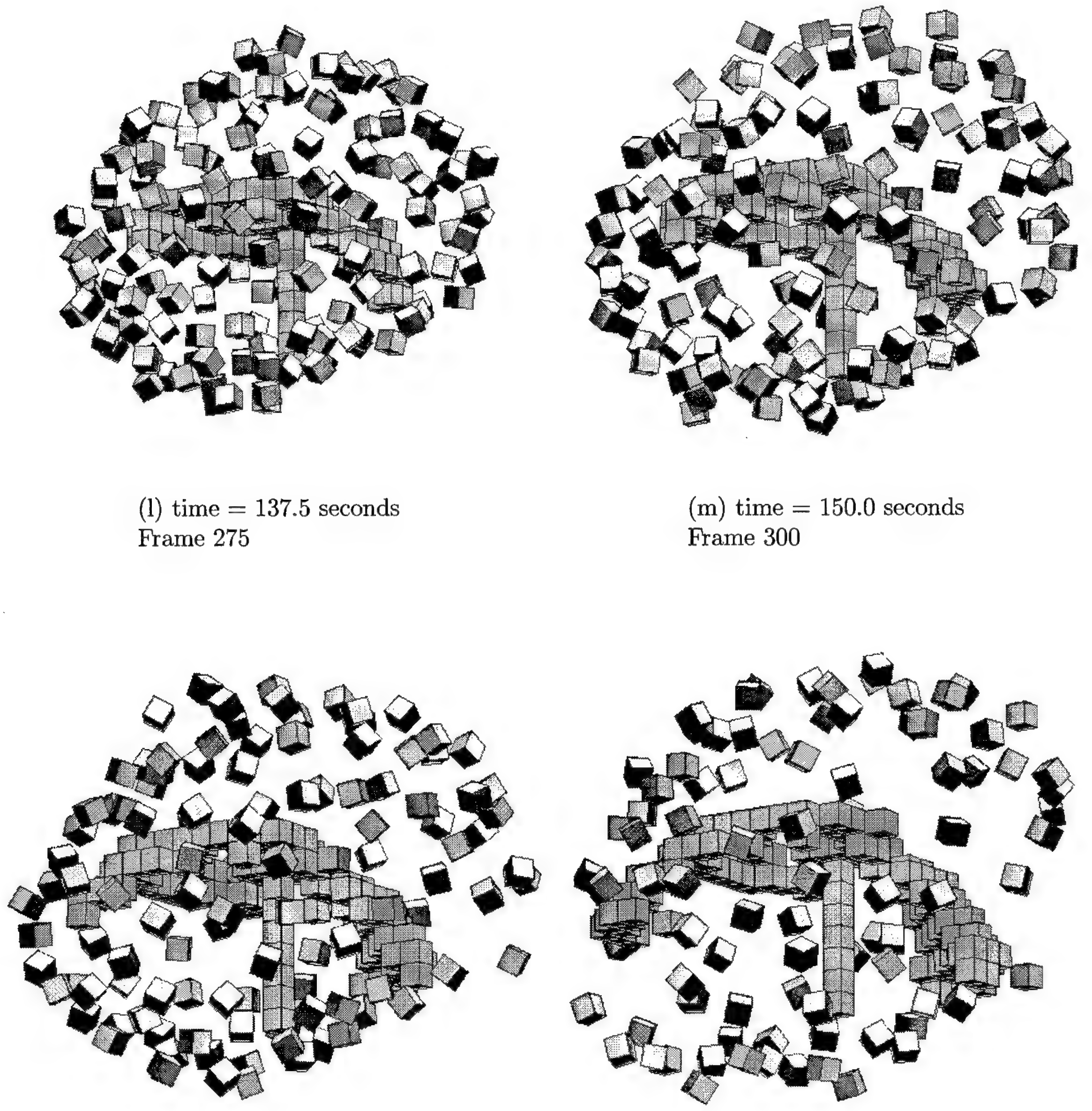
(i) time = 100.0 seconds
Frame 200

(j) time = 113.5 seconds
Frame 225

(k) time = 125.0 seconds
Frame 250

Figure 5.41 [h,i,j,k]

Case Study G: [page 3] *359 frames, 231 satellite constellation with system noise* $\sigma_{xyz}^2 = 0.0040$



(l) time = 137.5 seconds
Frame 275

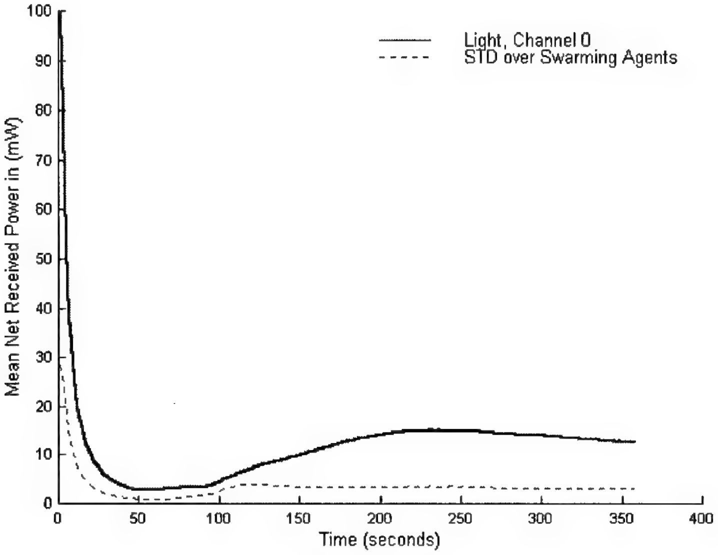
(m) time = 150.0 seconds
Frame 300

(n) time = 162.5 seconds
Frame 325

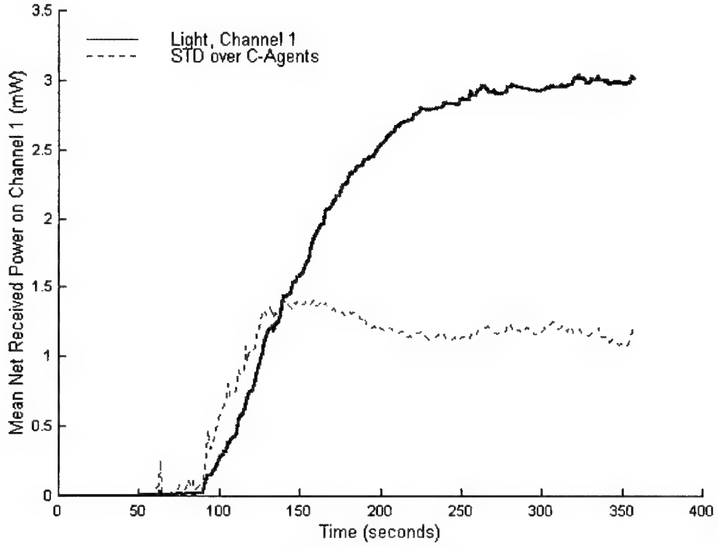
(o) time = 179.5 seconds
Frame 359

Figure 5.42 [l,m,n,o]

Case Study G*: Plots five and six of eight, Tranceiver CH-0 and CH-1..*



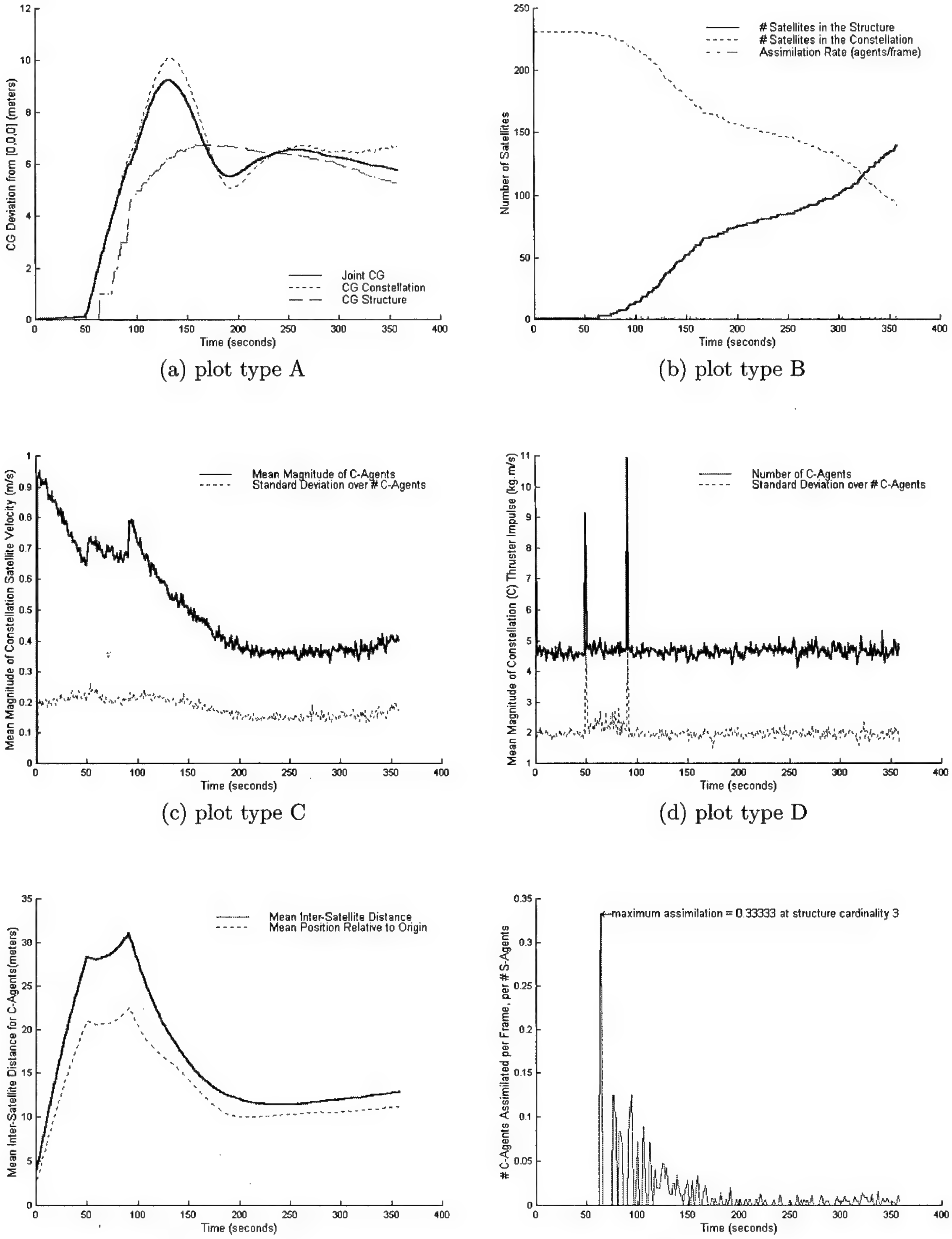
(e) plot type E



(f) plot type F

Figure 5.43 [e,f] Channel 0 and Channel 1 received power as a function of time

Battery G*: plots 1,2,3,4,7, and 8 of 8* *System Noise* $\sigma_{xyz}^2 = 0.0040$



(a) plot type A

(b) plot type B

(c) plot type C

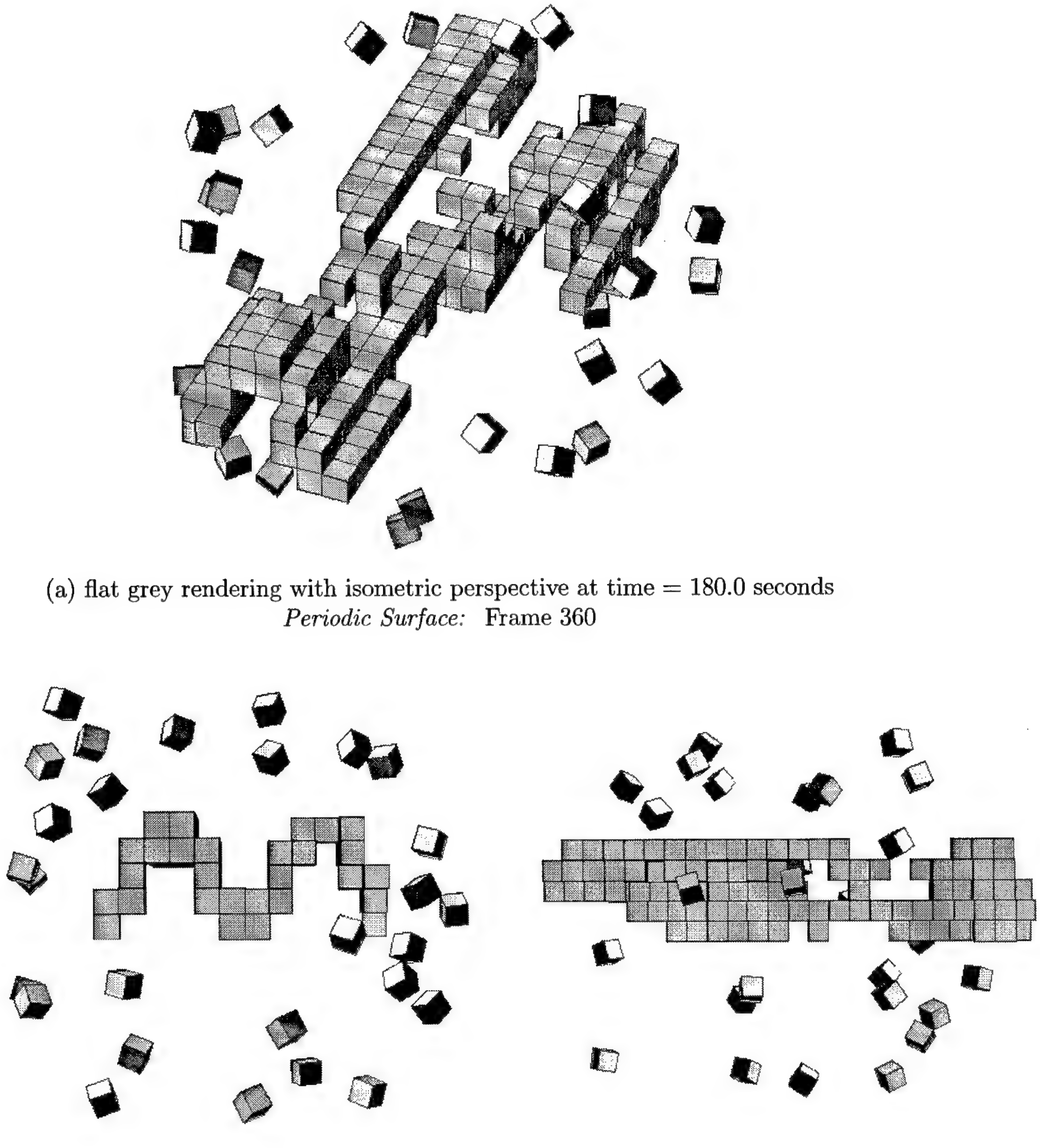
(d) plot type D

(g) plot type G

(h) plot type H

Figure 5.44 Battery G plots [a,b,c,d,g,h]

Case Study H: *360 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0040$



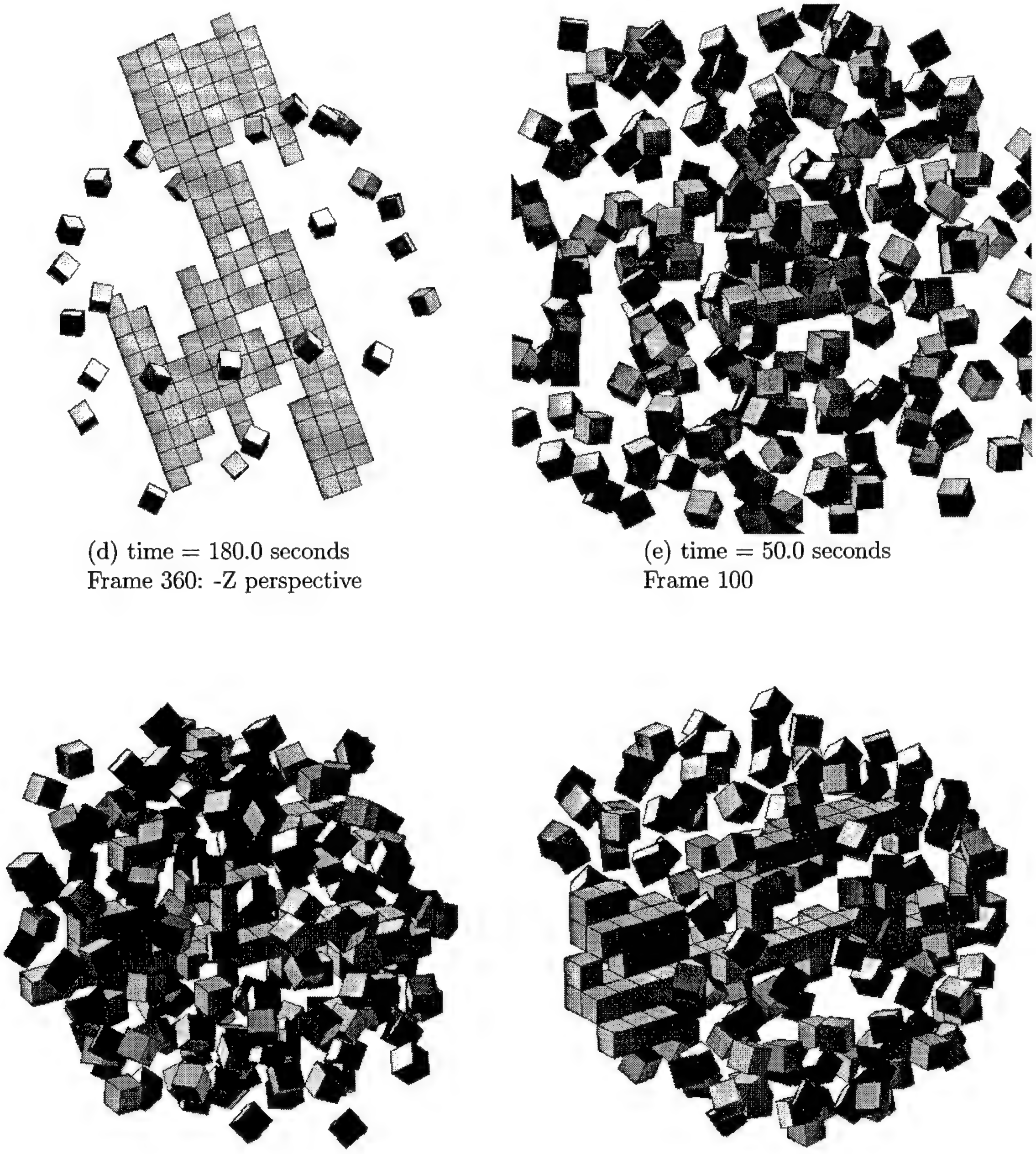
(a) flat grey rendering with isometric perspective at time = 180.0 seconds
Periodic Surface: Frame 360

(b) *Periodic Surface* [right view]
Frame 360

(c) *Periodic Surface* [front view]
Frame 360

Figure 5.45 [a,b,c] with low system noise at the agent level

Case Study H: [page 2] *360 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0040$



(d) time = 180.0 seconds
Frame 360: -Z perspective

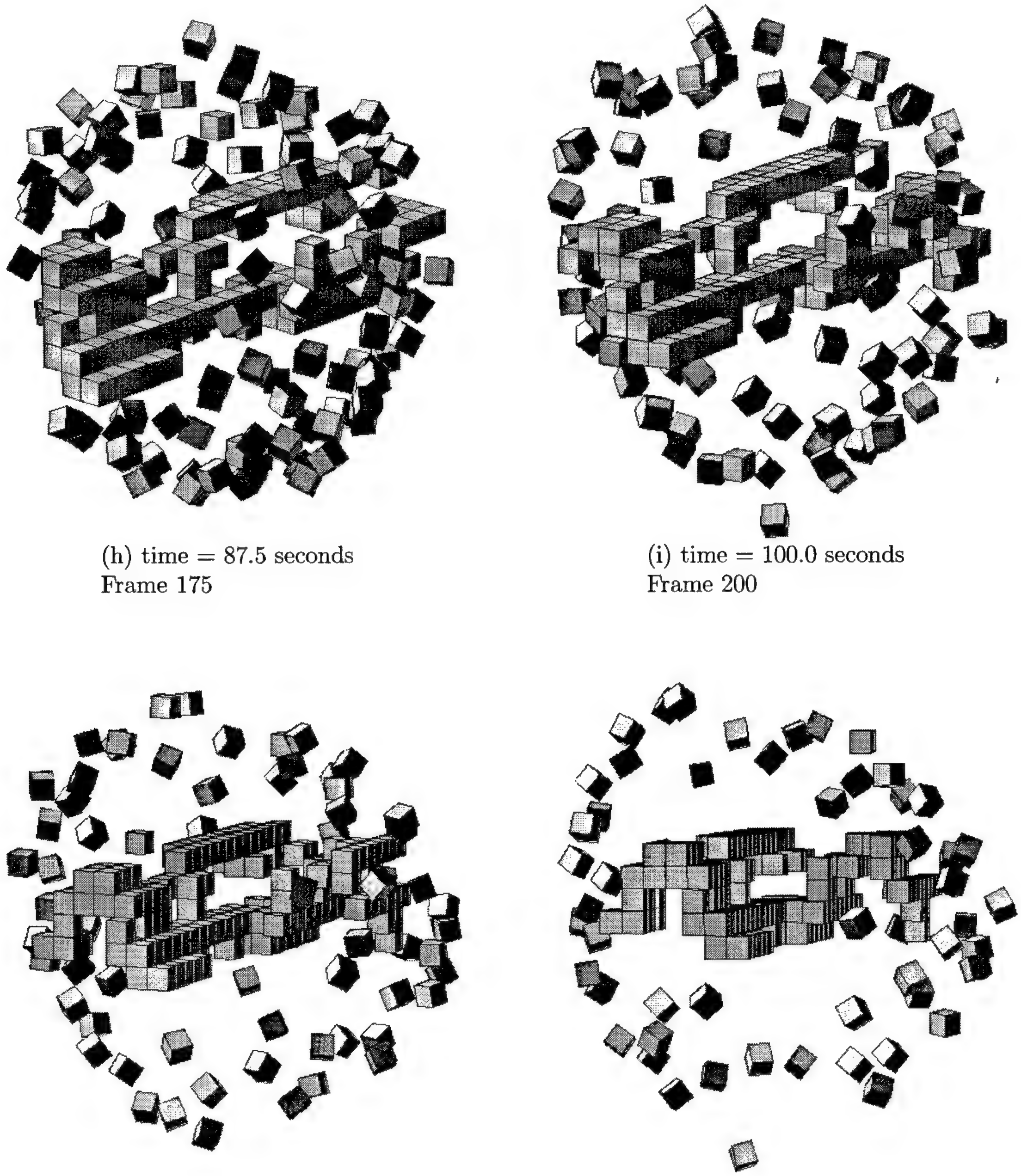
(e) time = 50.0 seconds
Frame 100

(f) time = 67.5 seconds
Frame 125

(g) time = 75.0 seconds
Frame 150

Figure 5.46 [d,e,f,g]

Case Study H*:* [page 3] *360 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0040$



(h) time = 87.5 seconds
Frame 175

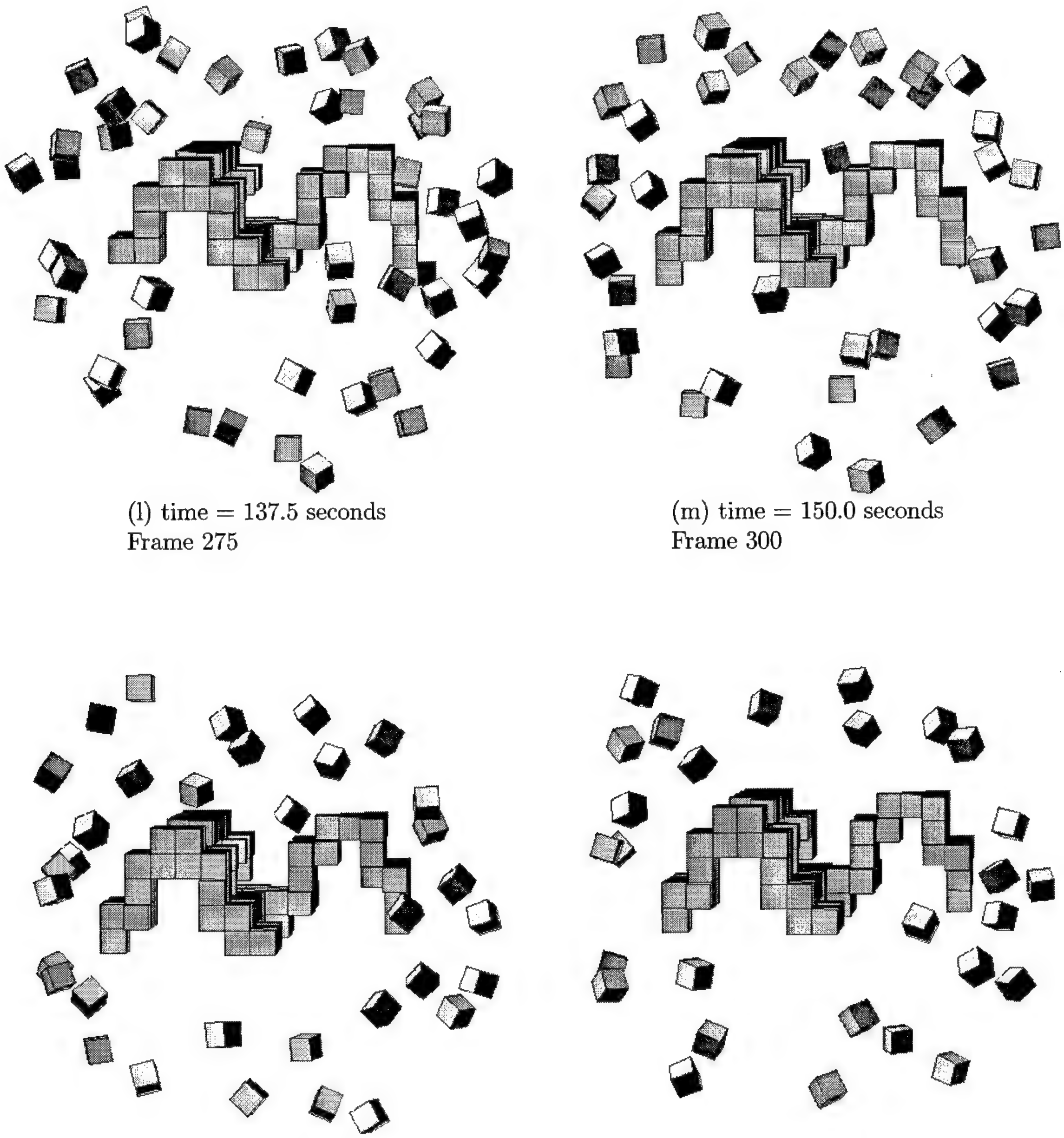
(i) time = 100.0 seconds
Frame 200

(j) time = 113.5 seconds
Frame 225

(k) time = 125.0 seconds
Frame 250

Figure 5.47 [h,i,j,k]

Case Study H*:* [page 4] *360 frames, 231 satellite constellation with system noise* $\sigma^2_{xyz} = 0.0040$



(l) time = 137.5 seconds
Frame 275

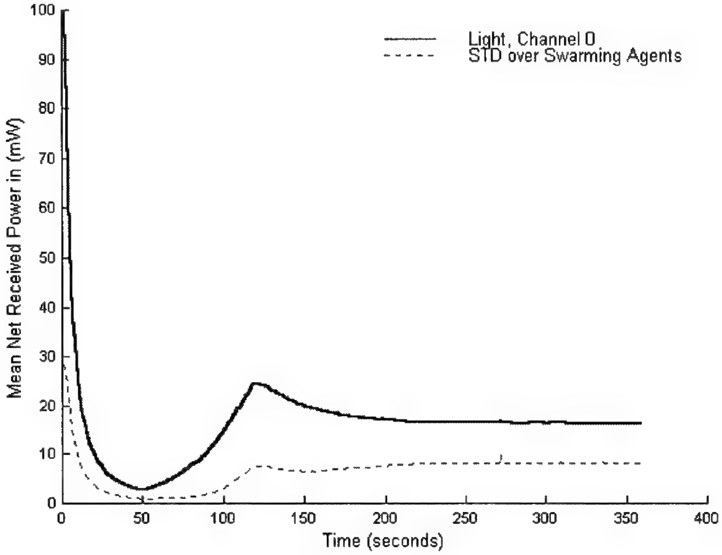
(m) time = 150.0 seconds
Frame 300

(n) time = 162.5 seconds
Frame 325

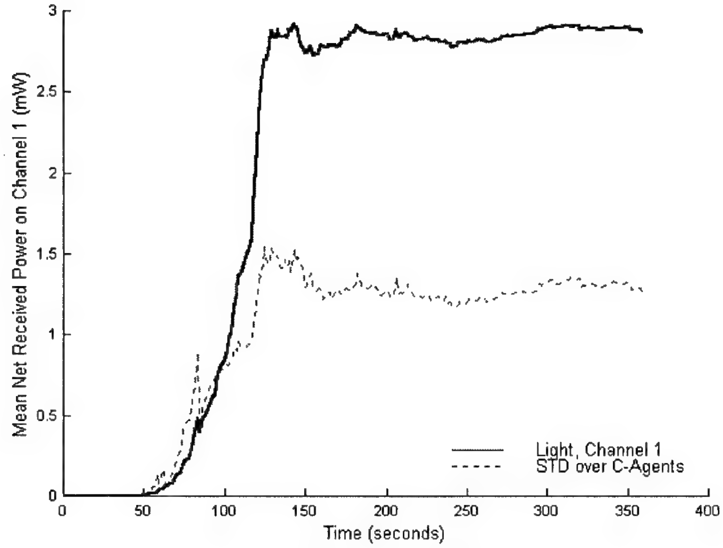
(o) time = 180.0 seconds
Frame 360

Figure 5.48 [l,m,n,o]

Case Study H*: Plots five and six of eight, Tranceiver CH-0 and CH-1.*



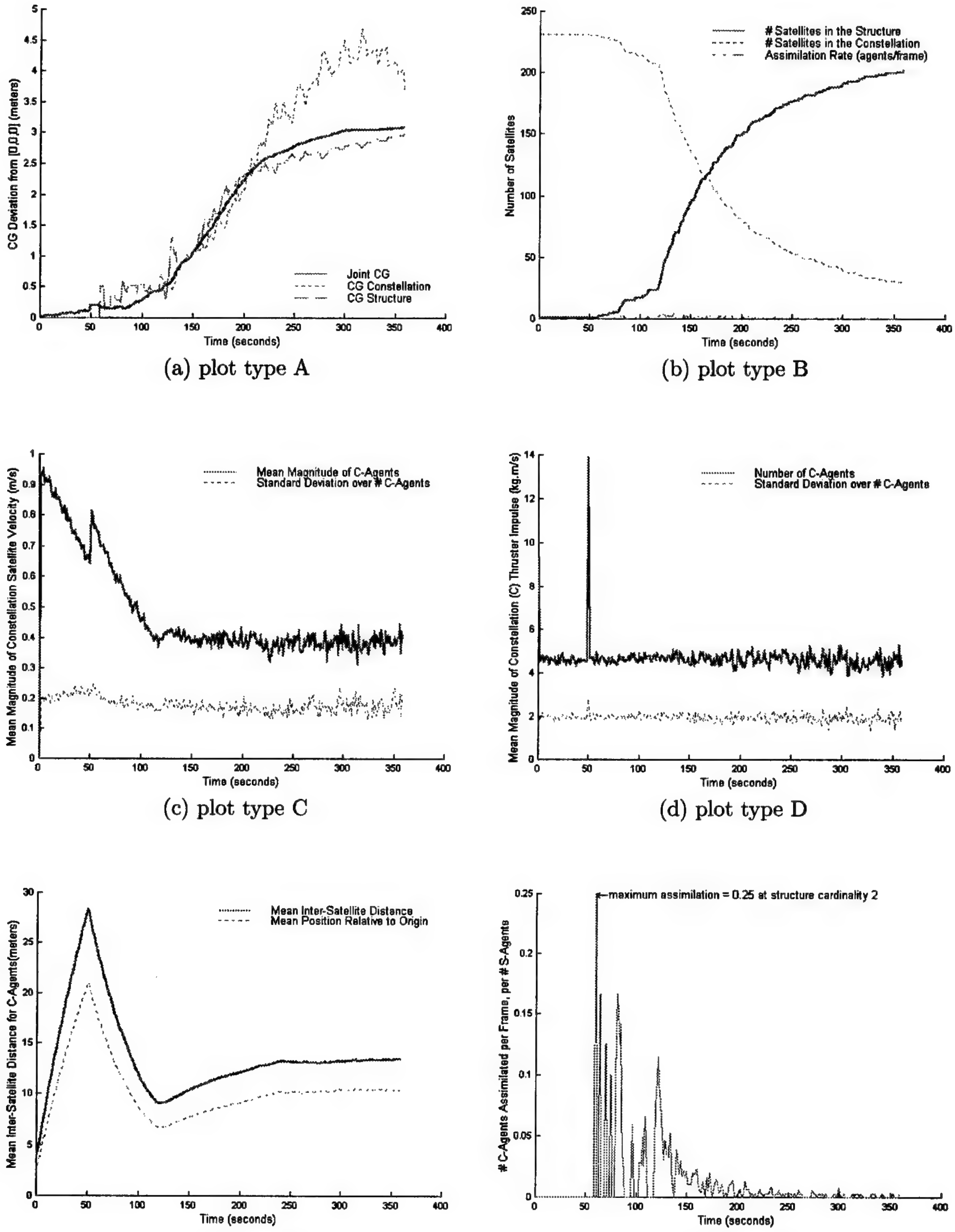
(e) plot type E



(f) plot type F

Figure 5.49 [e,f] Channel 0 and Channel 1 received power as a function of time

Battery H: *plots 1,2,3,4,7, and 8 of 8* *System Noise* $\sigma^2_{xyz} = 0.0040$



(a) plot type A

(b) plot type B

(c) plot type C

(d) plot type D

(g) plot type G

(h) plot type H

Figure 5.50 Battery H plots [a,b,c,d,g,h]

5.5 Analysis

5.5.1 Introduction

Figures (5.2(a) - 5.48(o)) span eight different simulations of four unique structures at two separate system noise levels. The initial configuration is a (4,7):231 payload, recall the convention (<radius>,<height>):<# agents>. Each of the eight simulations ends at 351 to 360 frames with 0.5 seconds between frames. Structure formation rates averaged 1.10 satellites per second from initial payload release. However, this rate is a function of the number of remaining satellites in the constellation. Compression forces applied during early structural formation have a lesser magnitude with fewer swarming agents. The concept of surface tension (discussed in chapter 3 accounts) for this interesting rate phenomena.

Consider the concept of high and low noise as applied to nano-satellite swarming. What is meant by 'high' and 'low' depends on capabilities of an agent. Low noise level is selected with a standard deviation of 5% of the maximum possible change in speed of an agent. The cap set on peak available impulse at any given time is 10 N·s; dividing by a nominal agent mass of 10 kg, yields a maximum possible change in speed of 1 m/s. Thus the low noise level defines an uncertainty in velocity of 5 cm/s (standard deviation). Similarly, high system noise is set to a standard deviation of 20 cm/s. This difference has definite effects on the results.

The set of eight simulations {A,B,C,D,E,F,G, and H} is divided into a low noise set {A,B,C,D} and a high system noise set {E,F,G,H}. The pairs {{A,E}, {B,F}, {C,G}, and {D,H}} account for identical architecture styles of the types {AE = half-cylinder, BF = paraboloid, GC = inverted paraboloid, and DH = periodic}. These pairs may be compared for noise effects, where the differing low noise sets allow comparisons between architectural styles. General deductions regarding satellite performance and swarming behavior may be made by examining the data set as a whole.

Mean impulse thruster data (kg·$\frac{m}{s}$) over two noise levels and four architecture types					
Case Study	# Frames	Primary	Secondary	Median Thrust	Arch. Type
A	351	07.1	8.5	1.2	inv. half-cyl
B	353	13.1	–	0.7	paraboloid
C	354	07.1	9.3	1.2	inv. paraboloid
D	355	13.2	–	0.7	periodic surf.
E	356	08.5	10.1	4.5	inf. half-cyl
F	358	13.9	–	4.5	paraboloid
G	359	09.1	11.0	4.7	inv. paraboloid
H	360	14.0	–	4.5	periodic surf

Standard Deviation	Rebound *Half-Cyl/Inv Parab.*		No Rebound *Parab./Periodic*		Median Thrust
	Prim.	Sec.	Prim.	Sec.	
0.0025	7.1	8.9	–	–	1.2
	–	–	13.15	DNE	0.7
0.0040	8.8	10.85	–	–	4.6
	–	–	13.95	DNE	4.5

Table 5.3 Mean impulse thruster data over two noise levels and four architecture styles

The paper document limitation is especially unfortunate for this thesis. Representing three-dimensional scenes that span time and color on a two-dimensional black and white surface (the printed page) is archaic at best. Thus use of perspective and multiple time-series plots are used extensively to convey the dimensionality of the scene; however, information is lost in the translation that may never be gained without experiencing a holographic movie of the results. Hundreds of simulations were generated for this document, but eight case studies required 43 pages of text. An ever increasing quantity of thesis data prompted the Air Force Institute of Technology to offer the option of a compact disk attachment to a thesis. A CD is available with movies and the entire set of data collected during the course of this work.

5.5.2 Increased System Noise

5.5.2.1 Half-Cylinder. As with other structures, the center of gravity relative to the Collective Inertial Reference (CIR) frame origin remains near zero from the payload break until the first structural member is placed at Γ_1. The duration of this interval is 25.0 seconds as prescribed by the state-transition uplink table outlined in chapter 2. After Δ_{12} is initiated, the joint CG begins to deviate at a rate of 0.8 m/s. This deviation oscillates as the satellites collapse from equilibrium due to a change in the balance of CH-0 and CH-1 forces. In the case of the half-cylinder, the deviation remains constant after 50% of the constellation satellites are assimilated.

We define the 'constellation half-life' as time from transition Γ_0, until the number of constellation satellites equals the number of structural satellites. Specifically,

$$ch \;=\; (t^{s=c} - t^0) \tag{24}$$

where $t^{s=c}$ is the time at equal constellation/structure membership and t^0 is the time of the first impulse firing . For the half-cylinder, ch = 103.0 seconds for the low noise case and 170.0 seconds for the high noise case. This result suggests that noise increases the structural formation rate. However, results from following structure types suggest a correlation between structure architecture and construction rate only. A conclusive relationship between noise levels and construction rate cannot be inferred from the available data.

From plots C and D of case study A (CS-A), it is apparent that even the mean velocities over all constellation satellites are not smooth functions of time. Distinct transitions evident in all case studies are apparent even at times not induced by structural seeding or payload release. Most apparent are the series of dual spikes on plot D of CS-A. The third spike in mean impulse thrust is due to the rebound after equilibrium collapse. The general sequence of events is an explosion, a brief (low magnitude) recoil, then a rapid collapse for structural assimilation. After the rapid collapse a second intense rebound occurs, signified by plot D, CS-A spike #3. Agent velocities then decrease to a constellation-wide average of 0.3 m/s. The minimum of this average is lower for CS-A (low noise) than for CS-E (high noise). Here, CS-E illustrates a mean satellite velocity of 0.4 m/s from t = 75.0 seconds to t = 180.0 seconds, whereas CS-A fluctuates from a low of 0.30 m/s to 0.33 m/s during the same time period. This result indicates an increased mean satellite velocity for constellations that experience higher system noise. The origin of this result is discussed in the general deductions section.

5.5.2.2 Paraboloid. If we compare the paraboloid to the half-cylinder from a noise standpoint, several influences are apparent. First, the half-lives (*ch*) for CS-F and H are 75.0 seconds and 78.0 seconds, respectively. The system with higher noise takes 3 seconds more to reach the same structural point. This margin is too low to draw a sound conclusion regarding the correlation between noise levels and construction rate. However, the half-cylinder half-life is between 103.0 and 170.0 seconds compared to 75.0 and 78.0 seconds for the paraboloid. This rate is nearly double and suggests a definite correlation between structure architecture and construction rate.

One additional comparison between CS-AE and CS-BF: if we look at plot type G for both case studies, we notice that the mean inter-satellite distance (dark line) peaks in CS-BF and then drops sharply, whereas CS-AE plateaus for 14.0 seconds before collapsing. This result is due to the elastic surface tension property, or in the case of CS-AE, a lack thereof. A single cusp on the mean inter-satellite distance curve suggests that structural seeding is initiated prior to expansion rebound. From an efficiency standpoint this is a most important discovery. Eliminating an equilibrium rebound (as a result of surface tension elasticity) by early seeding effectively conserves the quantity of fuel that a second rebound costs the system. The average duration of a rebound is 3 impulse firings. Furthermore, the magnitude of the mean impulse spike is 14 kg·m/s on average. To clarify, 'mean' generally refers to a mean over the constellation satellites and 'average' refers to an average of these mean values, but over architecture type or noise. Also, mean and average are equivalent, mathematically. Thus an average fuel savings of 38% is realized over the half-life duration.

5.5.2.3 Inverted Paraboloid. The inverted paraboloid (CS-CG) confirms the hypothesis that energy expenditure is directly proportional to system noise. It also provides evidence (see plot type B) that noise increases the construction half-life of a system. The *ch* of CS-G is 160.0 seconds, whereas the *ch* of CS-C is 113.0 seconds. However, the approximation of a paraboloid in CS-G is much different than the approximation constructed by CS-C. Even a small difference in geometry early on can lead to differing construction half-lives, which is the more probable scenario. As a quantitative metric does not exist for structural construction complexity, a corresponding definite answer to the architecture vs. noise relationship to *ch* remains elusive.

5.5.2.4 Periodic Surface. The periodic surface (CS-DH) presents a good example of a single, sharp transition from social behavior to constructive behavior as evidenced by the sharp peak in CS-D (plot type G). The mean satellite velocity peaks at 0.9 m/s during the initial payload burst. Velocities decrease, with sharp and periodic increases at state transitions, to a minimum of 0.2 m/s before gently increasing to 0.29 m/s at t = 3 minutes. This solution is typical for every constellation. The standard deviation (STDEV) of mean satellite speed is an important indicator of 'like satellite velocities' If every satellite in the constellation is traveling at precisely the same velocity, then the standard deviation about the mean is zero. As expected, satellite velocities are nearly equivalent immediately after the initial payload burst at 0.5 seconds, and CG-D and H plot type C supports this assumption. The standard deviation (plotted relative to the independent axis) increases from 0.05 m/s at t = 0.5 to 0.16 at t = 25.0 seconds (50 model seconds). This result implies that as satellites travel radially outward and their velocities decrease, they begin to travel at differing velocities. When state transitions occur, very deterministic actions take place and the standard deviation drops. When structural seeding occurs at 50 model seconds, the standard deviation of mean satellite speed decreases, which is true for every case study.

5.5.2.5 Transducer Receiver Power Statistics. Examination of CS-A and E for the half-cylinder and plot types E and F, which correspond to mean statistics for channel-0 and channel-1 reveals a trend. Plot type E (channel-0) for CS-E levels off at 18 mW of received power, whereas plot type E for CS-A levels off at 15 mW. Similarly, plot type F (channel-1) for CS-E levels off at 1.8 mWatts and plot type F for CS-A levels off at 2.7 mW of received power. The mean received power on these different channels provides a sense of the environmental factors to which the constellation satellites respond. The constellation in case study E is receives more power on social channels than construction channels than is the case in case study A. Since every satellite reaches equilibrium using the same binary behavior algorithm, these social forces are correlated to structure architecture only. The conclusion evident in these results supports the postulate that increased noise levels correspond to increased construction rates due to an annealing effect which breaks strong equilibrium (Chapter 2). It supports this postulate is supported because the low noise case (CS-A) has a higher structural force power level (2.7 mW) than the high noise case (1.9 mW). From a probability standpoint, CS-A is more likely to encounter structural forces strong enough to cause assimilation than CS-E. The satellite experiencing high dynamics noise (CS-E) spends more time avoiding collisions, as evidenced by a higher social power reception, then a low-noise counterpart. CS-BF,GC, and AE provide similar results.

Receiver power on channel-0 is greatest when transmitters are originally turned on at t = 0.5 seconds to induce swarming behavior. This result is evident from mean satellite velocity (plot type C), where on the plots satellite velocity is a maximum at t = 0.5 seconds due to the strong social repulsive forces induced by the sigmoidal binary behavior algorithm. It is possible to classify a constellation as either *elastic rebound* or *pre-equilibrium seed* by looking at the channel-0 curve near the Γ_1 transition at 25.0 seconds (50 model seconds). If the curve forms a shallow 'V' near 50 model seconds, then it is an early seeding variety and is more efficient from an energy expenditure standpoint. If the curve drops from 100 mW and then levels off at 50 model seconds for a short period of time before increasing, then the constellation is classified 'elastic rebound' (which experiences three major thruster burns instead of two).

5.5.2.6 General Deductions. The emergence of several notable types of behavior are evident when the entire data set is analyzed. The first notable behavior characteristic is coined 'equilibrium rebounding.' Rapid expansion from the payload configuration to a spherical constellation in equilibrium is halted by the effects of surface tension elasticity.

These effects are best described by analogy. Imagine a balloon covered with evenly spaced black dots. The dots represent satellites on the outer surface of a spherical satellite constellation. As air is blown into the balloon, the distance between these dots increases uniformly in every direction (in the same manner as our expanding universe). However, the behavior instilled in this constellation by the binary behavior algorithm can create a sensation of loneliness. When power levels below a certain threshold are received by a satellite, the binary behavior algorithm suggests to the control system that the next direction of travel be toward the brightest source of light. Avoidance outputs from the binary behavior algorithm are positive. Such outputs induce movement *away* from bright sources of light on CH-0. A rebound occurs when this uniformly expanding set of surface satellites coasts beyond the point of loneliness and contracts slightly from a fully expanded configuration. This contraction occurs almost instantly, since constellation synchronous thruster firing occurs every 0.5 seconds. The increase in satellite velocity due to this contraction is 60 cm/s on average and only occurs when structural seeding takes place after the point of maximal expansion is reached.

The effect of noise on the magnitude of thruster firings is apparent in plot type D for CS-(A-H). The mean magnitude of thruster impulse over the high noise structures (CS-A,B,C,D) is 4.6 N·s with a standard deviation of 2.0 N·s. For the low noise structures (CS-A,B,C,D) the mean magnitude of thruster impulse is 1.2 N·s with a standard deviation of 0.5 N·s. This result is significant for two non-trivial reasons. First, satellite impulse magnitude remains at a constant level during the mission, except during state transitions. Second, the correlation between system noise and thruster impulse magnitude is apparent: the ratios of induced noise levels to resultant impulse outputs are similar (4.6 N·s / 1.2 N·s) ≈ (0.20 STDEV / 0.5 STDEV).

This result suggests that the expended energy of a motive satellite constellation is linearly proportional to the noise levels present in the system. From the sensors to the thrusters, noise accumulates and ultimately induces an energy cost. We can imagine a small cluster of satellites moving relative to each other, yet accomplishing nothing, because noise is present in their inter-satellite distance estimates. The solution to this problem is simple: do not fire thrusters below the mean thruster magnitude level set directly by environmental noise. This solution may be executed adaptively by recording the magnitude and direction of thruster firings over time and estimating the associated probability distribution. The mean of this histogram is the lower bound for an efficient thrust range and requests for impulse firing below this floor should be denied. This solution creates an interesting hysteresis effect: it is possible for a satellite to drift until action is required in one direction, then to drift in the other direction until action is required, etc. Thus it is apparent that the probability of satellite collision is proportional to the system noise.

6. Conclusions and Recommendations

6.1 Introduction

The binary (swarming) behavioral algorithm and the four-post (structure formation) algorithm are shown to facilitate structural reconfiguration. A satellite constellation may be launched in solid form and reconfigured, via a 'gaseous' phase change, into a different pre-designed solid structure. The four-post algorithm is shown to efficiently convert global information, in the form of single valued functions, to local behavioral rules. Thus maximizing survivability and decreasing overall system cost. A trade-off in implementing the four-post algorithm is a decrease in the number of possible architectural styles.

6.2 Principles of Structure Driven Collaborative Behavior

Significant general descriptors related to collaborative satellite behavior as it applies to the structural formation mission are introduced. The first concept is *surface tension elasticity.* A constellation of agents held in close proximity, then released to seek equilibrium (using the binary behavior algorithm) expand from a central point. A spherical constellation forms naturally due to the binary behavioral algorithm, then expands beyond the equilibrium point due to inertia. The strong cohesive force of surface agents constrains constellation expansion and a rebound occurs.

The second metric introduced is 'constellation half-life.' The number of constellation satellites decreases as a decaying exponential function of time. The constellation half-life (ch) is that time when the number of structrual agents equals the number of constellation agents. It provides a quantitative metric with which to measure the structure formation rate. It also privides a time interval over which averages, such as the mean fuel-consumption, may be computed.

The third concept introduced is 'assimilation rebound.' Structural seeding occurs when agent-zero flies to the center of the constellation and initiates structure formation. The balance of channel-0 and channel-1 social forces is altered and the result is an implosion from social equilibrium (induced by channel-0 only) and the central assimilation zone. After constellation half-life is reached, the number of structural agents approaches an asymptote. Inter-satellite distance continues to decrease at a rapid linear rate; however, the number of active structural faces decays as an exponential function of time The result is a constellation rebound where mean inter-satellite distances increase and then plateau.

6.3 Summary of Results

A relationship between environmental noise levels and construction half-life is determined and a corresponding correlation between architectural style and construction rate is also proposed.

Eight unique architectures are constructed at two system noise levels. A four-fold increase in the uncertainty of satellite position induces a four-fold increase in the quantity of fuel required to reach constellation half-life. Thus, establishing a lower bound on satellite thruster firing, based on the median environment noise levels, decreases fuel consumption by 45%. Structural seeding prior to an equilibrium rebound (due to surface tension elasticity) reduces fuel consumption by an average of 38%. This significant decrease in fuel consumption is due to the inefficient second constellation-wide burn when agent-0 initiates structure formation after constellation equilibrium is reached. This method of fuel reduction is termed 'early seed fuel reduction.'

A direct correlation between architectural complexity and the construction rate is evident. The mean half-life for the half-cylinder and inverted paraboloid architectures is 234 frames or 1 minute 57 seconds. The mean half-life for the paraboloid and periodic structures is 155 frames or 1 minute 18 seconds. Comparing the two construction rates suggests that the simpler paraboloid and periodic structure architectures form 33.3% faster on average than the more complex half-cylinder and inverted paraboloid.

A noise variance of $\sigma^2 = 0.0040$ (compared to $\sigma^2 = 0.0025$) is shown to decrease the construction rate by 7.2%. This result indicates that a noise variance (for this particular implementation) that minimizes constellation half-life is between these two figures.

6.4 Recommendations for Future Research

The field of Distributed Satellite Systems (DSS) [2] was initiated in the 1990s by the Air Force Research Laboratory (previously Phillips Labs) Space Vehicles Directorate. This initiative is known as TechSat 21 and its mission is to demonstrate the feesibility of satellite formation flying. The AFRL is currently working with ten leading educational institutions to develope DSS technology. Ten contracts were awarded for the construction of formation flying satellite constellations to be launched by the year 2001. The Shuttle Hitchhiker palate is (currently) the most probable mode of transit into LEO.

Planned research up to the year 2007 focuses on formation flying in earth orbit. Orbital dynamics pose a number of obstacles and opportunities: certain orbits provide natural formation flying, i.e., the inter-satellite distances remain relatively constant over time, whereas other orbits require propulsive maneuvers to maintain relative positioning. Ideas ranging from inter-satellite tethers to solid (damped) tethers are now in the conceptual stage. It is inevitable that the transition from formation flying to structure formation must be addressed from a practical rather than a theoretical standpoint.

By the year 2020 questions posed and answered in this research should be addressed in the orbital environment *and* on earth. A body in zero gravity is capable of moving in the same manner as a neutrally buoyant object in water (or any fluid). Although this research emphasizes implementation in a frictionless three-dimensional environment, the friction induced by a viscous fluid simplifies many assumptions required for zero gravity environment (e.g., an agent speed limit). In zero-gravity, inter-satellite velocities can exceed the ability of a satellite to avoid collisions. Drag is directly proportional to the square of a bodies' velocity in a fluid. This effect creates a natural terminal velocity or speed limit and, hence, agents may be designed to withstand direct collisions without damage. Therefore, an environment with friction may be used to increase the probability of mission success by limiting the top speed of worker class agents. Structures may be formed under-water by identical autonomous [21,22] structural agents (elements). These structures may be nano-meters or kilometers on a side; the study of collaborative behavior imposes no scale constraints.

DSS is a new field, and a large number of possible variables may be explored. The limitations of computational power and time place debilitating constraints on the ability of the research reported here to discover more than a few fundamental principles that may be applied to future work. The effects of noise and architectural complexity answered many questions and proposed many more. Future research should increase the complexity of the binary behavioral model and the four-post algorithm so that coherent structure formation is realized. A C++ implementation of the MATLAB code is also required so that thousands or millions of satellites may begin to be modeled. However, current computer technology limits STEMS to constellations in the 300-500 satellite range, and the massive quantity of data stored for analysis is also prohibitive.

Several changes should be made to the STEMS model. First, structure rotation should be modeled with complete accuracy. As satellites attach, they must impart rotational and linear momentum on the orbiting structure. The result is a uniform structural stirring action and thus uniform structure formation; long stems or holes are less likely to form. Second, a death condition should be imposed. In this work satellites continued to activate faces until constellation resources were exhausted. Resulting in lopsided structure formation with holes. The addition of a death condition would indirectly allow selection of the payload size to completely account for every element in the resultant structure. Thus the edges of a paraboloid would stop growing based on some local knowledge, and the unfinished sides would then attract remaining satellites. The final suggestion is a focus on efficiency and the power budget. The net energy expended to construct a given structure can be minimized. Since the number of structural reconfigurations is a function of system energy and since certain environments devoid of light or heat require that agents use on-board power for propulsion the mission fails if power is exhausted prior to structure completion. Also, minimizing power requirements ultimately implies reduced cost in space applications.

The short term focus (1999 - 2006) should be on the formation flying problem [1,2,18]. However, the type of formation flying now being developed is headed in the wrong direction. Current algorithm development is focusing on precise positioning using the GPS. An un-interrupted reception of GPS signals must not be assumed. A focus on GPS positioning limits the environment for satellite constellations to earth orbit. Thus, years of research may be required to transition earth-reliant constellations to purely autonomous and self-sufficient constellations in orbit about other distant massive bodies or to terrestrial applications related by analogy. Algorithm development that relies on precise positioning and the transmission of constellation-wide ephemerid lists should also be avoided: global information passing immediately reduces both simplicity and robustness, and such methodologies are doomed to failure for spatially distributed constellations with millions of satellites. Motion scheduling has never been a topic of group discussion, (i.e., excessive intersatellite communication) in nature and it should not be for distributed satellite systems.

Appendix A. - Model Assumptions

A.1 Introduction

Operational parameters related to the *Structural Emergence Simulator (STEMS)* are addressed in the following sections. A model is only as good as its supporting assumptions and can only be made better by decreasing the margin between reality and the model. There are several ways to decrease this margin: one is to validate all assumptions as being closely correlated to existing systems or values and another is to introduce error with the proper distribution. If the error or assumptions are only slightly different from the actual system, then the model may be completely invalid over a large range of inputs. To decrease the probability of invalidating the model, we test its susceptibility to varying inputs. If the results vary dramatically from results for the actual system, then steps should be taken to remedy the poor assumptions.

All assumptions made in this thesis are explained and validated to the best extent possible in the following sections. The precision of MATLAB comes into question on a number of occasions. On each occasion, the precision of MATLAB is at least 9 orders of magnitude *better* than required that allow rapid iterative routines in place of 'absolute' routines that require re-calculation of large matrices.

Consider the problem of rotating a body in space. A rotation matrix **R** is required, which rotates a body **B** about some vector $\overrightarrow{a}$ over an angle $\Delta\theta$. Repeated rotations of **B** are done in one of two ways. Either **B** is updated by rotating over an absolute angle θ a number N times, thus saving (N-1) re-calculations of $\mathbf{R}_\theta$, or **B** is updated by repeatedly rotating by the same $\mathbf{R}_{\Delta\theta}$. Much computation time is saved by adopting the latter methodology.

MATLAB precision can be employed creatively to decrease computation time and maximize the *quantity* of experimental results without a corresponding sacrifice in *quality*.

A.2 Matlab Precision Example: Iterative Rotation Validation

Consider a free-floating agent in space. We model the body frame of this agent by a matrix $\mathbf{B}$ (N x 3), with a Center of Gravity (CG) at the rotation point [0,0,0]. The matrix $\mathbf{B}$ is comprised of vectors $\mathbf{B} = (\overrightarrow{v}_1, \overrightarrow{v}_2, ... \overrightarrow{v}_N)$ and it rotates relative to a given vector $\overrightarrow{a}$ about its CG with some angular velocity (w). This rotation is modeled by viewing the body at discrete time intervals $\{\overrightarrow{t}_1, \overrightarrow{t}_2, ... \overrightarrow{t}_\Omega : \Delta t = (t_{n+1} - t_n)\}$. Each time the body is updated from one time to the next a rotation must take place given that $w \neq 0$ and $\overrightarrow{a}$ exists. One question is; do we rotate based upon an absolute stored angle, thus re-calculating the rotation matrix $\mathbf{R}_\theta$ for each angle, or do we use the previous matrix $\mathbf{B}$ in conjunction with the previous rotation matrix $\mathbf{R}_{\Delta\theta}$ to update the scene, thus eliminating repeated re-calculation of $\mathbf{R}_\theta$? The answer depends on the cumulative error associated with the latter methodology. A detailed analysis is performed, and the iterative method is validated.

A function 'mcad2.m' is developed which returns a (3 x 3) rotation matrix $\mathbf{R}$ when passed both a rotation axis $\overrightarrow{a}$ and a rotation angle θ (in radians). In code this relationship is [R] = mcad2($\overrightarrow{a}$, θ).m. An analysis answers the question: 'over how many successive rotations is the error (in position) negligible compared to the existing noise?' Noise is introduced by solar wind, non-ideal thrusters, and other impending forces that affect rotational velocity. Figure (2 [upper]) demonstrates that the average distance between the tips of two randomly selected vectors; one rotated by 2π and the other rotated 3,000 times by $\frac{2\pi}{3000}$ is on the order of $2{\cdot}10^{-13}$ units out of one with a standard deviation (STDEV) of $1.4{\cdot}10^{-13}$. This result suggests that an arbitrary body can be rotated through space using the rotation vector $\mathbf{R}$ repeatedly using the methodology $\mathbf{B}^+ = (\mathbf{B}^-)(\mathbf{R}_{\Delta\theta})$ provided that w is constant. Figure (2 [lower]) confirms this hypothesis and reveals a linear relationship between the number of repeated rotations about $\Delta\theta$, N, and the cumulative error in position. Notice that the error STDEV increases according to the linear equation STDEV(N).

$$ERROR(N) \approx 6.4686 \cdot 10^{-15} \cdot N - 2.5163 \cdot 10^{-13} \quad (1)$$

$$STDEV(N) \approx 4.7837 \cdot 10^{-15} \cdot N - 3.5309 \cdot 10^{-13} \quad (2)$$

Using '**R-Recycling**' error in position accumulates linearly according to Error(N) and we note that *Error(10,000 repetitions)* $= 6.44{\cdot}10^{-11}$ parts per meter.

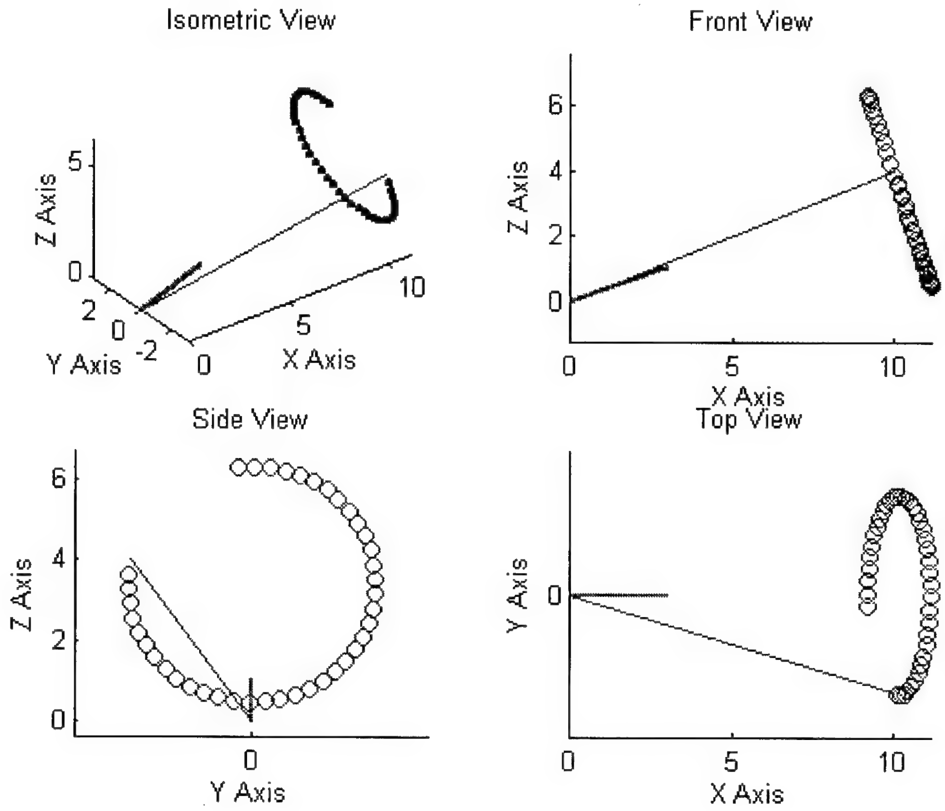
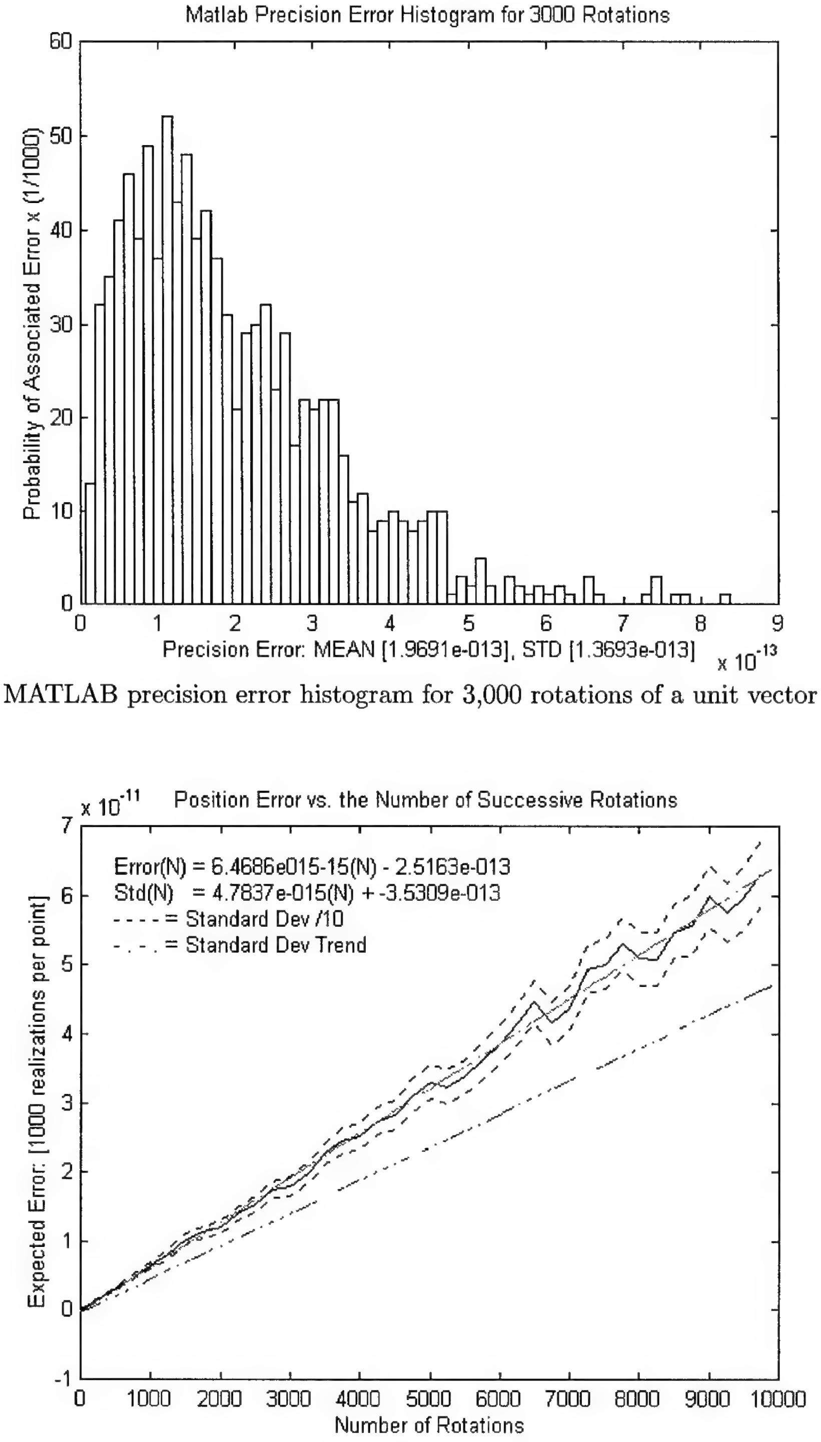


Figure 6.1 Rotation of an arbitrary vector about a different arbitrary vector A.



MATLAB precision error histogram for 3,000 rotations of a unit vector

Unit vector tip-position error vs. number of successive rotations

Figure 6.2 MATLAB precision error analysis.

Figure (2) is the result of 780,080,000 rotations of a unit vector about evenly distributed realizations of $\overrightarrow{a}$. Each point on the error line (solid line) is the expected value of 2,000 random vector selections and each was rotated N times by an angle $\frac{N}{2\pi}$ about $\overrightarrow{a}$. The process 'test_mcad.m' took 9 hours and 18 minutes of processor time (on an Intel Pentium II® 200MMX, 64 MB RAM) and validated the hypothesis that $\mathbf{B}^{+} = \mathbf{B}^{-}(\mathbf{R}_{\Delta\theta})$**,** or that **'R-Recycling,'** is a *very acceptable* updating method.

A.3 The Agent

A.3.1 Uniform Mass distribution

We seek an agent with no preferential direction in an air, water, or space environment. Furthermore, we seek an agent with rotation characteristics about the exact center of a cube to simplify the control system and associated optical and electromagnetic sensory inputs. Later it is determined that physical symmetry and even mass distribution yield significant simplifications in the behavior of agents and the associated modeling code. Unless we intend to make use of the gravity gradient present in orbit or the force of solar wind, the agents remain balanced about a center point.

A.3.2 Uniform Orientation Probability

Consider a right hexahedron floating in an environment with a homogeneous set of forces acting upon it. It is symmetric about 12 planes and is not granted *apriori* orientation knowledge relative to the surrounding cluster. The satellite does have a three-axis inertial reference frame that can be relied upon until precession error accumulates excessively. Also, it has a uniform orientation probability because it does not seek to align itself with the given coordinate system. In space the choice of coordinate system is aligned with the existing gravity gradient, which has no alignment effect on a satellite with even mass distribution.

A.4 Inter-Agent Relationships

A.4.1 Foundress Specialization

All foundress (i.e., worker class) satellites are of equal mass. Further research may be performed in which the mass distribution varies according to:

$$\delta_i^m = [m_0 + K_i] \quad where \quad K_i \backsim \mathcal{N}(0, \sigma_m^2) \tag{3}$$

Given a set range of possible thrust magnitudes, a difference in mass changes the 'handling characteristics' of a given agent. The acceleration range is inversely proportional to the thrust range, so a large σ_m^2 may result in agents incapable of keeping up with the swarm during distance travel.

A.4.2 Line-of-Sight Communication

Line-of-sight (LS) communication in a collective environment can serve two purposes. First, a break in data transfer suggests (to the receiver) that the transmitter was shadowed by a satellite or structure of greater priority so that transmitter data priority is partially determined by accidental shadowing. Second, solid LS communication grants the satellite *apriori* information that a (current) flight path exists between the transmitter and the receiver. This information may not exist for long, but can be used in conjunction with distance and/or Doppler information to optimize the receiver trajectory.

A.4.3 Non Line-of-Sight Radio Communication

Any number of low frequency bands are suitable for broadcast radio communication intended for the group as a whole. Broadcast communication is used for synchronization, leadership assignment, and parameter upgrades. It is assumed that every satellite in the cluster and structure receives nearly 100% of all broadcast messages. Note: there is a finite probability that one or more satellites will suffer from catastrophic bit error. Electromagnetic communication error rates increase with noise or attenuation and decrease with additional signal power, error correction hardware, and/or bandwidth. If a satellite should fail to receive a critical broadcast message, then it is considered a *mutation.*

A.4.4 Mean shadow approximation

In certain circumstances it is beneficial to approximate the shadow of a cube with that of a sphere. This approximation is valid in the far field and helps to simplify the simulation code considerably. When optical communication devices are employed, shadowing is a concern. If one agent is much closer than another within a small angle, then near agents can shadow far agents, effectively cutting off communication. But how do we determine the average shape of a shadow cast by a cube? The answer is to sum the ensemble of all two dimensional shadows cast by a cube of width w that is rotated to arbitrary and evenly distributed orientations.

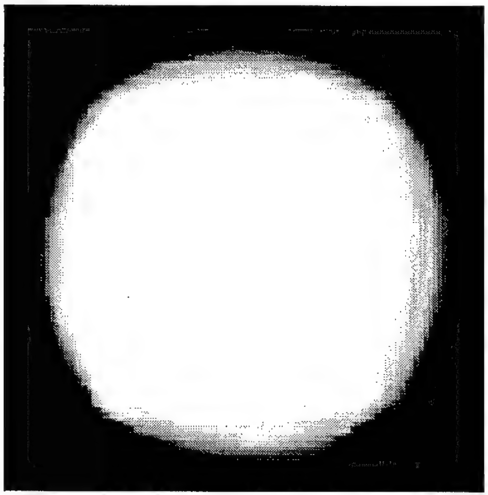
Thus, a cube of width w is rotated to 817 unique and uniformly distributed orientations. A light source along the Z-axis (of infinite distance from the cube) casts shadows on the XY plane (Figure 3 [top]) below the cube at position [0,0,0]. These shadows are captured as a binary image in the MATLAB function 'shadow.m' and summed as an ensemble. The summation is normalized between 0 and 1 and now represents a probability distribution $\rho_s(i,j)$ (Figure 3 [bottom]), where i and j represent Cartesian Coordinates on the X-Y plane. We know beforehand that a circle of radius $R_1 = \frac{w}{2}$ will be uniformly shadowed with probability 1. We also know that the region outside of a circle of radius

$$R_2 = \frac{w\sqrt{3}}{2} \tag{4}$$

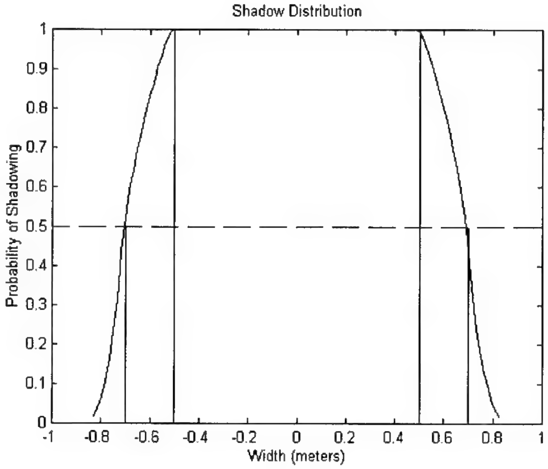
will have no shadow cast upon it. The purpose of this analysis is to find the radius of the shadow cast 50% of the time. It is not clear that the shadow cast 50% of the time is given by

$$\overline{r}(w) \;=\; \frac{1}{2}\left(\frac{w}{2}+\frac{(R_1+R_2)}{2}\right)=\frac{w(\sqrt{3}+1)}{4}\approx 0.68301 \;\; \text{for} \;\; \text{w=1} \qquad (5)$$

However, experimental results estimate $\overline{r}$ = 0.6976. This figure deviates 2% from the mathematically derived hypothesis. Thus hypothesis is assumed true and the spherical approximation of a cube is Equation (5).



Ensemble shadow of a 1 x 1 x 1 cube



Unit vector tip-position error vs. number of successive rotations

Figure 6.3 Probability of complete shadowing.

Appendix B. - STEMS MATLAB 5.0 Functions

B.1 function [runn,fasteph] = act_init(agent)

```
%————————————————————————————————
% function [dt,M,runn] = (agent)
% Purpose: to gather necessary data from available sources (uis,agent)
% and output variables required for the run sequence.
%—- INPUT VARIABLES:
%agent: / (structure) with all initial conditions information
%—- OUTPUT VARIABLES:
%fasteph:/ (structure) with dt,M,N,D,RP,Po,Pn,Tn,Rn,Bn,Sn matrices
%runn: / (structure) with enough information to render a sequence
%—- INTERNAL VARIABLES:
%N: /the number of satellites in the 'agent' structure
%————————————————————————————————
global rui hui
dt = ev(hui(12));                    % Get time steps from 'dt' edit box
M = ev(hui(11));                    % Get number of frames from 'M' edit box
N = size(agent.phys,2);              % Number of satellites

% RUNN IS IDENTICAL TO AGENT, EXCEPT FOR THE FRAMING
runn.disp = agent.disp;              % General information is identical to 'agent struc-
ture'
runn.frame(1).phys = agent.phys;
% — FRAME STRUCTURE 'RUNN' —
% A default configuration for the RUNN structure is defined
% ( ) implies an abreviation for the structure name
% # implies a user defined variable. The rest are defaults
% Major Class: RUNN (agent)
% 1. display (disp)
%# a) color (1 x 3) :a basic printable display color
% b) vertices (8 x 3)(vert) :the corners of a cube
% c) tvert (6 x 3) :the sensor vectors
% d) imap (K x K) :image map maps onto faces for great visuals
% g) cmap (K x 3) :colormap
%# e) dbit (1 x 1) :decides display mode
% f) fac (6 x 4) :corner connect matrix
% 2. frame(k).physical (phys) % FRAME TERM ADDS TIME DIMENSIONALITY
%# a) position (1 x 3)(p) :where the agent is located
%# b) trajectory (1 x 3)(t) :where the agent is going
%# c) rotation (1 x 3)(r) :how the agent is rotating
%# d) mass (1 x 1)(m) :how massive the agent is
%# e) Rinertia (1 x 1)(i) :the rotational inertia of an agent
% f) body (8 x 3)(b) : a rotated version of 'vert'
% g) tvector (6 x 3)(s) : a rotated version of 'tvect'
```

B.2 function act_sensors(m);

```
% Purpose: To generate a new version of RP (Received power)
% in 'fasteph'
N = fasteph.N; % number of satellites
k_xmit = fasteph.kx; % transmission gain
k_rec = fasteph.kr; % reception gain
agent_old = runn.frame(m); % ephemerides
for agent_ID = 1:N % Find the incident power on each satellite
[incident_power] = incident(agent_old, agent_ID, N, k_xmit, k_rec)
RP(agent_ID) = incident_power; % Make an (N x 6) power matrix
end % Sum the number of shadows cast
fasteph.RP = RP;
```

B.3 function activate

```
global runn rui hui
[action] = udtovar(rui(1)); % Green/Red playback buttons
switch(action)
case 1 % Activate_I : Initial run sequence with calculation
fprintf('************************************************************')
fprintf('%s\n','Running activate_I: Swarm Calculation in Progress')
[runn] = activate_I; % _I = Initial Calculation
case 2 % Activate_R : Playback from saved 'runn' file; type 'filename_r'
[lower] = udtovar(hui(15)); % Lower frame limit
[upper] = udtovar(hui(16)); % Upper frame limit
fprintf('%s\n','Running activate_R: Swarm is Rendered from an "_r" file')
activate_R(lower,upper); % _R = Render from global memory ('runn')
case 3 % Activate_M : Play a saved movie; type 'filename_m'
% CURRENTLY THIS FUNCTION DOES NOT EXIST !!
[lower] = udtovar(hui(15)); % Lower frame limit
[upper] = udtovar(hui(16)); % Upper frame limit
fprintf('%s\n','Running activate_R: Swarm is Rendered from an "_r" file')
activate_R(lower,upper); % _R = Render from global memory ('runn')
otherwise
fprintf('%s\n','No valid case found in <activate.m>')
end
```

B.4 function [runn] = activate_I

```
% Activate Run-Time Function
% Purpose: the main loop in which one state of the agent
% is carried to the next state and so on. Time begins at zero,
% time is incremented by 'dt' and ends after M increments for a total time
% of total_time = ((M-1) x dt);
%
% During each iteration, all important data is stored in a structure array
% called 'runn' and it may be stored in the form of a sequence of GIFs if
% desired. Since movies can get extremely large, the GIF sequence may prove
% a more efficient way to store large runs.
% INTERNAL VARIABLES
%——— The following variables are elements of the structure ———
```

```
% dt :time interval between frames
% M :number of frames total
% N :the number of satellites
% D :Inter-Sat distance matrix (N x N)
% RP :(N x 6) received power matrix. N satellites, 6 faces
% P :(N x 3) new satellite positions
% T :(N x 3) new satellite trajectories
% R :(N x 3) new rotation vectors
% B :(8 x 3 x N) new body frames
% S :(6 x 3 x N) new sensor vectors
% runn :runn time structure. It holds all calculated information over M frames
% IT: (N x 3) Tangential Impulse (m/s) Not kgm/s!! (No need)
% IR: (N x 3) Rotational Impulse (in units of rotational intertia)
global agent runn hui color imap cmap fac...
d_bit SP STR NMAT RT0 move O3 O8 SBODY SSENS
%*** INITIAL PHYSICAL VARIABLES
M = ev(hui(11)); % Get number of frames from 'M' edit box
N = size(agent.phys,2); % (scalar) Number of satellites
dt = ev(hui(12)); % Get time steps from 'dt' edit box
xpower = agent.disp.xp; % (scalar) (Watts) Universal Transmitter Power
orate = (360/60)*(pi/180)*dt; % Rotation rate = 360 degrees / t seconds.
[RATE] = mcad2([.5 .5 .5],eye(3),pi,orate); % Structure rotation matrix.
STR = uint8(zeros(60,60,60)); % (60 x 60 x 60) structure configuration matrix
% in 'isat3' you can fiddle with this and rp0,1,2 stuff
mass = agent.phys(1).m; % (scalar) all agents have the same mass
stm = zeros(N,1);
radius = 3.0; % Sticking Radius
k_xmit = 1; % Transmitter Gain (absolute)
k_rec = 1; % Receiver Gain (absolute)
dR0 = zeros(N,1); % (N x 1) in(radians) Previous theta values
%*** INITIAL RENDERING VARIABLES
color = agent.disp.color; % (1 x 3) face color if dbit = 0
imap = agent.disp.imap; % (K x K) face image map if dbit = 1
cmap = agent.disp.cmap; % (K x 3) face color map if dbit = 1
fac = agent.disp.fac; % (6 x 4) Facial connection matrix
d_bit = agent.disp.dbit; % {0,1, or 2} and specifies rendering
%*** SPEED-UP VARIABLES (ALL ARE DECLARED GLOBAL)
[RT0] = make_RT0(N); % (3 x 3 x N) Null rotation matrices
%*** THE SATELLITE MOOD ZERO (Collective Behavior) PARAMETERS —
[mood0] = make_mood0(mass,xpower); % behavioral profiles
[mood1] = make_mood1(mass,radius,xpower); % behavioral profile
% — UPDATE MAIN GUI UICONTROLS —
set(hui(1),'Value',1); % Bring it back to the start
set(hui(1),'Min',1); % Set the minimum frame # on our slider
set(hui(1),'Max',M); % Set the maximum frame # on our slider
set(hui(22),'String',['dt:',num2str(dt)]); % Time increment
%///////// EFFECTIVE MAIN LOOP START \\\\\\\\
% — MAKE THE FIRST RUNN FRAME FROM AGENT —
[runn] = first_runn(agent,dt,xpower,N); % No globals
ric % Render the first frame
tic % Start StopWatch
for m = 2:M % MAIN LOOP (M TIMES)
```

```
% ————————- GATHER SENSORY DATA ————————
[P0,T0,R0,B0,S0,ECM,stm] = pull(runn.f(m-1)); % No globals
[NT,P12,D] = near_sats(P0); % No globals
[RP0,RP1,RP2,A] = sense(N,S0,ECM,NT,D,stm,k_xmit,k_rec,xpower,radius); % No
globals
%——————————————————————————
%— CALCULATE THE BEHAVIOR INDUCED DESIRED UPDATES —
% IN TANGENTIAL AND ROTATIONAL VELOCITY
seed = 5; % If seed == 0 then no seeding occures
% < global SBODY SSENS SP STR move > in behave1_b
[IT,IR,P0,B0,S0,ECM,stm] = behave1_b(seed,m,A,P0,T0,
R0,B0,S0,RP0,RP1,RP2,ECM,N,stm,mood0,mood1);
%—— UPDATE IN DIRECTION AND ROTATION —-
T0 = (T0 + IT); % (N X 3) Tangential
R0 = (R0 + IR); % (N X 3) Rotational
[SCG] = find_SCG; % Find the structural center of gravity
for q = 1:N % Effectively freezes structural members
if (stm(q)==1) % Operate on structural members
T0(q,:) = [0 0 0]; % Freeze trajectory motion
R0(q,:) = [0 0 0]; % Freeze rotation
else % Operate on swarming elements
P = (P0(q,:) - SCG); % Center CG on the origin
P = P*RATE; % Rotate about the origin
P0(q,:) = (P + SCG); % Re-center on the origin
end
end
%———- THE MATRIX VECT0R VERSION OF D = (R x T) IS DONE HERE ————
-
[dP] = t_update(T0,O3,dt); % Update in position (N x 3) in (meters)
[dR,AV] = r_update(R0,O3,dt); % Update theta (N x 1) in (rads)
[RT1] = rt_update(dR0,dR,AV,RT0,N); % Create Rotation matrices
% ———————————— TEST FOR TRACKING MODE ——————————
if (d_bit~=2), cla; end % Agent previous path tracing mode
% — DO INDIVIDUAL UPDATES T0 SATELLITES —-
runn.f(m-1).RP0 = RP0; % Received Power on channel 0
runn.f(m-1).RP1 = RP1; % Received Power on channel 1
runn.f(m-1).RP2 = RP2; % Received Power on channel 2
runn.f(m-1).IT = IT; % Translation Impulse Magnitude
runn.f(m-1).IR = IR; % Rotation Impluse Magnitude
runn.f(m).P = (P0 + dP); % Move according to dP
runn.f(m).T = T0; % Maintain speed
runn.f(m).R = R0; % Maintain angular velocity
runn.f(m).ECM = ECM; % Emitter Channel Matrix
runn.f(m).stm = stm; % Structure membership
% — RENDER ALL N SATELLITES IN FRAME m —
%set(gca,'CameraTarget',[0 0 0]); % Keeps the camera pointed at the Structural CG
[BODY,SENS] = render_main(m,RT1(:,:,1),B0(:,:,1),S0(:,:,1),runn.f(m).P(1,:),runn.f(m).P(1,:));
runn.f(m).B(:,:,1) = BODY; % New body frame
runn.f(m).S(:,:,1) = SENS; % New sensors frame
for i = 2:N % Only move swarming agents
[BODY,SENS] = render_main(m,RT1(:,:,i),B0(:,:,i),S0(:,:,i),runn.f(m).P(i,:),runn.f(m-
1).P(i,:));
```

```
runn.f(m).B(:,:,i) = BODY; % New body frame
runn.f(m).S(:,:,i) = SENS; % New sensors frame
end
hit_lights(1);
% —– UPDATE MAIN GUI EPHEMERIS INFORMATION —–
drawnow % Dump MATLAB graphics solution
t_elapsed = toc; % Get current real time
feedback(m,M,dt,t_elapsed);
end % ////// MAIN LOOP END \\\\\\
% WARNING! This function calls a GUI button.
overnight % used for overnight automatic saves of ”runn”
% — PUT ZEROS IN THE LAST RECEIVE POWER MATRIX—
runn.f(M).RP0 = zeros(N,6);
runn.f(M).RP1 = zeros(N,6);
runn.f(M).RP2 = zeros(N,6);
```

B.5 function activate_R(lower,upper)

```
% function activate_R(lower,upper)
% Purpose: to render satellites from a pre-calculated runn file
global agent runn hui fac color imap cmap d_bit
cla % Clear the screen regardless of what is on it
if isempty(runn), load foursats1_r.mat; end
M = size(runn.f,2); % Number of frames
if (upper > M)|(lower<1)|(lower>=upper)% Check to see if the range is valid
upper = M; lower = 1; % If not, then make it valid
vartoeud(1,hui(15)); % Put it in the editbox and userdata
vartoeud(M,hui(16)); % Put it in the editbox and userdata
end
mrange = (upper - lower + 1); % Number of frames to be played
dt = runn.dt; % Time increment
N = size(runn.f(1).P,1); % Number of satellites
wait = 0; % Real time speed register
O8 = ones(8,1); % Speed matrix
fac = agent.disp.fac; % (6 x 4) Facial connection matrix
color = agent.disp.color; % (1 x 3) face color if dbit = 0
imap = agent.disp.imap; % (K x K) face image map if dbit = 1
cmap = agent.disp.cmap; % (K x 3) face color map if dbit = 1
d_bit = agent.disp.dbit; % {0,1, or 2} and specifies rendering
ve(M,hui(11)); % Send frame range to the # Frames editbox
set(hui(1),'Value',lower); % Bring it back to the start
set(hui(1),'Min',lower); % Update the lower limit
set(hui(1),'Max',upper); % Update the upper limit
set(hui(12),'String',num2str(dt));% Put 'dt' in the dt editbox
tic; % Speed Tester for real time updating
for x = 1:550,dot(4,5);end % speed testing loop used dot product
ttt = toc; tfactor = (550/ttt); % (loops/sec)
for m = lower:upper % play over the range
tic; % Start stopwatch
set(gca,'CameraTarget',[0 0 0]); % Keeps the image rock steady with no shifting around
if (d_bit~=2), cla; end % Tracking render mode = 2
```

```
for i = 1:N
BODY = runn.f(m).B(:,:,i); % extract the body frame
CG1 = runn.f(m).P(i,:); % current CG
VP = (BODY + O8*CG1); % translated body
if (m==lower)
CG0 = runn.f(lower).P(i,:); % first CG
else
CG0 = runn.f(m-lower).P(i,:); % old CG
end
plot_sat(VP,CG0,CG1);
end
hit_lights(1);
% —— UPDATE MAIN GUI EPHEMERIS INFORMATION ——
set(hui(1),'Value',m); % Update the frame slider
set(hui(2),'String',[num2str((m-lower)*dt),' sec']); % Clock
set(hui(21),'String',[num2str(m),':',num2str(mrange)]);% Frame count
drawnow
tlag = toc'; % Time stretching operation to
des_time = 2;
if (tlag < des_time) % Put your desired frame time here
% if (tlag < dt)&(dt < .3) % slow down the machine a bit
clicks = (des_time-tlag)*tfactor'; % # loops required to slow down
for x = 1:clicks
dot(4,5);
end % keep it slowed down
end
tlag2 = toc;
set(hui(26),'String',['etf:',num2str(tlag2)]); % Put time on screen
end
```

B.6 function add_element(np,ce);

```
% Purpose: to add the ID # of a new structural element at position np
% offset by (30 30 30)
global STR
xp = (round(np) + [30 30 30]); % [0 0 0] goes to [30 30 30]
ce = uint8(round(ce)); % You can't be too safe
STR(xp(1),xp(2),xp(3)) = ce; % Assign the Agent_ID to the element
```

B.7 function [A1] = afilter(A0)

```
%Purpose: to make a list of all valid sticking assignments
A1 = [];
Aa = unique(A0(:,1)); % Floater sat ID
Ab = unique(A0(:,2)); % Structure sat ID
Ac = unique(A0(:,3)); % Face number
la = length(Aa);
lb = length(Ab);
lc = length(Ac);
```

```
c = 1;
for i = 1:lb
for j = 1:lc
if (c<=la)
A1 = [A1;[Aa(c) Ab(i) Ac(j)]];
c = (c + 1);
end
end
end
```

B.8 function [az,el] = azel_slider(handle1,handle2)

```
% function [az,el] = azel_slider(handle1,handle2)
% Purpose: to return azimuth and elevation in degrees from two sliders
% given by handle1 and handle2
% Elevation Range = -90 degrees to 90 degrees
% Azimuth Range = -360 to 360 degrees
el = get(handle1,'Value');
az = get(handle2,'Value');
```

B.9 function [IT,IR,P0,B0,S0,ECM,stm] = behave1_b(seed,m,A,P0,T0,...

```
R0,B0,S0,RP0,RP1,RP2,ECM,N,stm,mood0,mood1)
% function [IT,IR,P0,B0,S0,ECM,stm] = behave1_b(seed,A,P0,T0,R0,S0,RP0,RP1,RP2,N,stm,mood0,
% Purpose: to return two impulse vectors that facilitate
% movement toward a goal defined by Desired Agent Velocities
% given by (Ds0,Ds1, and Ds2).
% Ds0: "social" field
% Ds1: "growth" field
% Ds2: "at bay" field
% mood is a structure with elements
% mood.p = transmit power for every transmitter
% mood.cs = Safe collision radius
% mood.de = Equilibrium distance (radius)
% mood.a = Maximum attractive field force
% mood.r = Maximum repusive field force
% mood.imax = Effective velocity boost at thurster full power
% seed: 0 if you don't want to seed and between 1 and M if you do
% You can edit the mood parameters in 'activate_i.m'
global SBODY SSENS SP STR move
% — STRUCTURAL SEEDING OCCURS HERE —
if (m==seed) % Seed if frame m == seed
ee = 1; % First element ID #
stm(ee) = 1; % Declare agent #1 a structural agent
P0(ee,:) = [0 0 0]; % Position at origin
B0(:,:,ee) = SBODY; % Square body
S0(:,:,ee) = SSENS; % Square sensors
add_element(P0(ee,:),ee); % First structural position
SP = P0(ee,:); % The first structural element
```

```
[NBRS,flag] = local_sense(ee,P0(ee,:)); % Local neighbors and exclusion test
[ECM] = update_ecm(ECM,NBRS,P0(ee,:),ee);
end
% — POWER CONVERSION TO SINGLE VECTOR —
for i = 1:N
Power0 = RP0(i,:); % (1 x 6) Power Received on Channel 0
Power1 = RP1(i,:); % (1 x 6) Power Received on Channel 1
Sensors = S0(:,:,i); % (6 x 3) Sensor (unit) Vectors
W0(i,:) = Power0*Sensors; % (1 x 3) Effective Rec Pwr on Channel 0
W1(i,:) = Power1*Sensors; % (1 x 3) Effective Rec Pwr on Channel 1
end
% — FIELD TRANSFER FUNCTIONS DS0 AND DS1 —
[MW0,UW0] = n_vector(W0); % Watts = (N x 1) magnitude, Wnorm = (N x 3) normal
[MW1,UW1] = n_vector(W1); % Watts = (N x 1) magnitude, Wnorm = (N x 3) normal
[DS0] = p_field(MW0,UW0,mood0); % Desired Destination Velocity (m/s)
% — STRUCTURAL POSITION AND STICKING ROUTINE —-
if (m >= seed)
[DS1] = p_field(MW1,UW1,mood1); % Desired Destination Velocity (m/s)
if (~isempty(A))&(m >= seed) % If there are agents to subsume
noc_list = [0]; % This is a list of recently stuck agents
for j = 1:size(A,1) % Stick 2,3,4... per clock cycle
ce = A(j,1); % Cluster Element
if (length(find(noc_list==ce))==0) % ”ce is not in the noc_list”
se = A(j,2); % Structural Element
face_num = A(j,3); % Face number
np = (P0(se,:) + move(face_num,:)); % np = Requested position of cluster element
[NBRS,flag] = local_sense(ce,np); % Local neighbors and exclusion test
if (flag==1) % We are allowed to stick to the stucture!
noc_list = [noc_list ce]; % Add to the noc_list NEVER TO STICK AGAIN!
add_element(np,ce); % Add the latest member to our structure
SP = [SP;np]; % Used later for CG calculation
P0(ce,:) = np; % New Position of St
B0(:,:,ce) = SBODY; % Square body frame
S0(:,:,ce) = SSENS; % Square sensor frame
stm(ce) = 1; % Make cluster element officially a structural element
[ECM] = update_ecm(ECM,NBRS,np,ce);
fprintf('Frame: (%d) agent %d to side %d of satellite %d\n',m,ce,face_num,se)
fprintf('satellite %d now sits in position [%d,%d,%d]\n',ce,np(1),np(2),np(3))
end % Sticking routine
end
end % A-list entry
end % Empty test for A-list
else
[DS1] = zeros(N,3); % Until something is seeded, nothing is active
end
% — SUMMATION OF RESPONSIVE VECTORS AND ADDITIVE THRUSTER
NOISE —
DS = (DS0 + DS1); % Net desired velocity (m/s)
%IT = clip_vector((DS-T0),mood0.imax); % Clip IT off with magnitude (Iomax/mass)
IT = (DS-T0); % Unclipped solution
stdv = 0.05; % Standard Deviation in thruster error
ITN = randn(N,3)*(stdv); % 5 percent STD in position
```

```
IT = IT + ITN; % Add on the noise
IR = zeros(N,3);
```

B.10 *function [io] = clip_ds(i,imax)*

```
% function [Io] = f_impulse(I,Imax)
% Purpose: to take in the desired impulse I (m/s) and threshold it for
% values greater than Iomax. to return the viable impuse Io for satellite
% thrusting purposes
[Inorm] = n_vector(I); % Normalized I vector
Imag = norm(I); % Magnitude of the I vector
if (Imag > Iomax)
Iomag = Iomax; % Io = Iomax, Iomax < I < inf
else
Iomag = Imag; % Io = I, 0 < I <= Iomax
end
Io = Iomag*Inorm; % Resultant thrust vector in 3D
```

B.11 *function [vout] = clip_vector(vin,imax)*

```
% function [VOUT] = clip_vector(VIN,Imax)
% Purpose: to take in the desired impulse I (m/s) and threshold it for
% values greater than Iomax. to return the viable impuse Io for satellite
% thrusting purposes
O3 = ones(1,3);
[MV,UV]=n_vector(VIN); % Magnitude and direction of VIN
MV(find(MV>Imax))=Imax; % Every element of MV larger than Imax = Imax
VOUT = UV.*(MV*O3); % Return clipped impulse vectors
```

B.12 *script cube.m*

```
% ——————————————————————————————————————
% Purpose: Visualize a random distribution of cubes in space
% and test a set of required functions for 'main'
% ——————————————————————————————————————
% ——— START Screen Initialization ————
figure(1); close(1) % Initialize figure(1)
set(0,'units','pixels'); set(gcf,'units','pixels'); % level the playing field with pixels
ScreenSize = get(0,'ScreenSize'); % Aquires the resolution of the current screen
upper_x = ScreenSize(3); upper_y = ScreenSize(4); % resolution of the current monitor
left = upper_x/2; bottom = upper_y/2; % lower, left coordinates of figure window
width = left; height = bottom; % figure dimensions
set(gcf,'Position',[left (bottom-50) width height]) % page 99 and 103 [Marchand]
set(gcf,'Name','Modeling the Collective Behavior of Micro-Satellite Clusters')
set(gcf,'NumberTitle','off') % get rid of the numbered figure scheme
set(gcf,'MenuBar','none') % get rid of the stupid menu bar below the title
```

```
axes(’Position’,[0 0 1 1]) % fill the entire figure with the axes
set(gcf,’Color’,[0 0 0]) % [0 0 0] = black
%set(gcf,’Color’,[1 1 1])
% ——— END Screen Initialization ————
% ——— Initialize variables ——————-
pi2 = pi*2;
% Declare the root vertices
vert = [0 0 0;...
0 1 0;...
1 1 0;...
1 0 0;...
0 0 1;...
0 1 1;...
1 1 1;...
1 0 1];
vert = (vert - 0.5);
fac = [1 2 3 4;2 6 7 3;4 3 7 8; 1 5 8 4;1 2 6 5;5 6 7 8];
% — Initial Variables —
O = ones(8,1); %Later used to convert a (1 x 3) to an (8 x 3) with identical rows
degtorad = pi/180;
% Consider a Satellite Status Matrix ”S”
% S = [ X, Y, Z, Xr, Yr, Zr ] where ”r” refers to an angular rotation magnitude
% S = [ P , R ] where ”P” refers to the position matrix and ”R” rotation.
P1 = [0 0 0;1 0 0;2 0 0;3 0 0;0 0 -1;3 0 1]; % double ended L
P2 = [0 0 0;1 0 0;2 0 0;0 0 1;2 0 1;4 1 4]; % ’U’ shape
P3 = [0 0 0;0 0 1;0 1 0;0 1 1;0 2 0;0 2 1;1 0 0;1 0 1;1 1 0;1 2 0;1 2 1;4 1 4];
% 3 sided not possible orientation
[P4,A4,T4] = randomp(10,.5,0); % makes 10 satellites with random orientations
[P5,A5,T5] = randomp(200,.5,0); % makes 80 satellites
Pb = P5; % the position matrix collection
Ab = A5; % the rotation matrix collection
thetab = T5; % rotation angles in radians
ls = length(Pb); % the number of structural satellites
hold off
%’——— Updating routine ———
for n = 1:ls % done once for each cube
A = Ab(n,:); % get the rotation vector from Ab (ls x 3)
theta = thetab(n,1); % get the rotation angle from thetab (ls x 1)
P = Pb(n,:); % get the position from Pb (ls x 3)
[R]= mcad2(A,theta); % find the rotation matrix using ’A’ and ’theta’
Sat = O*P + vert*R; % vert is {N x 3}, C is {3 x 3}
B = [0 .4 1]; % the color of each cube is determined by B
% B = [1 1 1];
color = B; % set the color of satellite faces
patch(’faces’,fac,’vertices’,Sat,’FaceColor’,color);
% draw the cube!
end
light(’Position’,[20 20 20]);
material shiny
axis vis3d off
figure(1)
axis equal
```

```
%rotate3d
view([90,0])
%colorbar
movecamera(80)
```

B.13 function [h] = ehandle(localtag,default_value_in)

```
% function [h] = ehandle(localtag,value_in)
% INPUT VARIABLES
% localtag: must be a string of the form ['tagname']
% it is the Tag assigned to a UICONTROL
% default_value_in: must be a NUMBER not a string.
h = findobj('Tag',localtag);
set(h,'String',num2str(default_value_in));
set(h,'UserData',default_value_in);
```

B.14 function etoud(localtag)

```
% function etoud(localtag)
% Purpose: to take data out of an editbox and move it to
% UserData for later recovery
data = str2num(get(localtag,'String'));
set(localtag,'UserData',data);
```

B.15 function [variable] = ev(handle)

```
% function [variable] = ev(handle)
% Purpose: handle in, variable from 'String' out.
variable = str2num(get(handle,'String'));
```

B.16 function [Z] = f(X,Y)

```
% Purpose: to add the underlying behavior to the swarming satellite
Z = (1/10)*(X.^2 + Y.^2);
%Z = (1/10)*(X.^2 + Y.^2); % Parabola
%Z = ones(4,1); % Flat plane at level 3
```

B.17 function feedback(m,M,dt,t_elapsed)

```
global hui
set(hui(1),'Value',m); % Update Frame Slider
set(hui(2),'String',[num2str((m-1)*dt),' sec']); % Clock
set(hui(21),'String',[num2str(m),':',num2str(M)]); % Frame Count
set(hui(26),'String',['et:',num2str(t_elapsed)]); % Put time on screen
```

B.18 function [SCG] = find_SCG

```
%Purpose: To find the center of gravity of the structure
global SP
if ~isempty(SP)
SCG = mean(SP,1);
else
SCG = [0 0 0];
end
```

B.19 function [runn] = first_runn(agent,dt,xpower,N)

```
runn = [];
for i = 1:N
P(i,:) = agent.phys(i).p; % Position (N x 3)
T(i,:) = agent.phys(i).t; % Trajectory (N x 3)
R(i,:) = agent.phys(i).r; % Rotational Velocity (N x 3)
B(:,:,i) = agent.phys(i).b; % Body frame (8 x 3 x N)
S(:,:,i) = agent.phys(i).s; % Sensor vectors (6 x 3 x N)
RP0(i,:) = agent.phys(i).trans.rp0; % Received power channel 0 (N x 6)
RP1(i,:) = agent.phys(i).trans.rp1; % Received power channel 1 (N x 6)
RP2(i,:) = agent.phys(i).trans.rp2; % Received power channel 1 (N x 6)
ECM(i,:) = agent.phys(i).trans.ecm; % Emitter Channel Matrix (N x 6)
stm(1,i) = agent.phys(i).stm; % Structural Membership (1 x N) (binary)
end
IT = zeros(N,3); % Translation Impulse Magnitude
IR = zeros(N,3); % Rotation Impluse Magnitude
RUNN = struct(’P’,P,’T’,T,’R’,R,’B’,B,’S’,S,...
’RP0’,RP0,’RP1’,RP1,’RP2’,RP2,’ECM’,ECM,’IT’,IT,’IR’,IR,’stm’,stm);
runn = struct(’dt’,dt,’xp’,xpower,’f’,RUNN); % Make a runn.dt field
```

B.20 script flyby.m

```
%FLYBY IS A COMPLEX ROUTINE USED TO VISUALIZE A 3D OBJECT OVER A
PRE-CONFIGURED
%TIME AND SPACE ROUTINE. THIS ROUTINE IS DEFINED BY THE MATRIX ’F’
figure(1) % Call the current figure out to the foreground
D = 29666; % The length of one movie frame
%N = the number of frames in the movie
%Azlow = lower Azimuth starting point : Azhi = upper Azimuth ending point
%Ellow = lower Elevation limit : Elhi = upper Elevation limit
F1 = [20,0,360,30,30,80,80]; %The 360 flyby at 30 degrees elevation
F2 = [20,0,180,-90,90,70,70]; %Upward spiral 360 flyby
F3 = [40,0,180,30,30,50,150]; %Zoom through the swarm
F4 = [40,0,90,50,50,80,80]; %High and tight but short twist
F = F4; %Tell the program which flyby routine to run F1, F2, F3 ...
AzimA = (F(3)-F(2))/F(1); AzimB = F(2); % Converts flyby routines to linear equation
coeffs
ElevA = (F(4)-F(5))/F(1); ElevB = F(5); % ” ”
```

```
ZoomA = (F(7)-F(6))/F(1); ZoomB = F(6); % " "
N = F(1); % Extract the number of iterations from the routine (F)
axis equal % Squares everything up
set(gca,'NextPlot','replacechildren') % Effecively makes it so that each updated figure
% is the same size
set(gca,'CameraViewAngleMode','manual') % Keeps MATLAB from automatically scaling the view
% each time
set(gca,'Projection','perspective') % Makes things further from the camera smaller
M = zeros(29666,N); % Dimensions the Movie matrix (M)
for i=1:N % N iterations
Az = i*AzimA + AzimB; % Linear Azimuth equation
El = i*ElevA + ElevB; % Linear Elevation equation
view([Az,El]) % Variable passing
movecamera(i*ZoomA+ZoomB) % Move camera in or out
M(:,i)= getframe(1); % Capture frame and store in the movie matrix (M)
end
figure(2) % Bring up a new figure for the video
set(gcf,'Color',[0 0 0]) % Set the figure color
axes('position',[0 0 1 1]) % Make the axes equal in size to the figure
movie(M,10) % Run the movie N times
close(1)
```

B.21 function [ce] = get_element(np);

```
% Purpose: to add the ID # of a new structural element at position np
% offset by (30 30 30)
global STR
xp = (round(np) + [30 30 30]); % Convert cartesian to matrix coords
ce = STR(xp(1),xp(2),xp(3)); % Find out who is in position xp
ce = double(ce); % Convert it to a double
```

B.22 function gicf(action)

```
% function gicf(action)
% INPUT VARIABLES
% action: Selects the appropriate callback based upon the button pushed
% OUTPUT VARIABLES
% none
% INTERNAL VARIABLES
global agent gic hui
switch(action)
case 'g1'
S = get(gic(1),'UserData');
set(gic(1),'String',num2str(S));
case 'g3'
etoud(gic(3)); % String to UserData
case 'g2' % Handles Slider Bar Fine Zoom Control
etoud(gic(2)); % String to UserData
```

```
vartoeud(1,gic(23)); vartoeud(1,gic(24)); vartoeud(1,gic(25));
% RANDOM Results Generator: Also home for arrays stored in UserData
case ’g4’ % Random Ephemeris Generator
gicf_a(5,gic(2),gic(4),gic(23),gic(6),gic(7),gic(8)); %30 meter cube
case ’g10’ % Random Ephemeris Generator
gicf_a(2,gic(2),gic(10),gic(24),gic(12),gic(13),gic(14)); % +/- 1 meters/sec
case ’g16’ % Random Ephemeris Generator
gicf_a(2,gic(2),gic(16),gic(25),gic(17),gic(18),gic(19)); % +/- 1 radian/sec
% — LAST Column Generator
case ’g5’ % Last Position Function
gicf_b(gic(2),gic(4),gic(23),gic(6),gic(7),gic(8));
case ’g11’ % Last Position Function
gicf_b(gic(2),gic(10),gic(24),gic(12),gic(13),gic(14));
case ’g26’ % Last Position Function
gicf_b(gic(2),gic(16),gic(25),gic(17),gic(18),gic(19));
% — X Column Changes
case ’g6’ % Change an element of an Ephemeris matrix
gicf_c(gic(2),gic(4),gic(23),gic(6),gic(7),gic(8),1);
case ’g12’ % Change an element of an Ephemeris matrix
gicf_c(gic(2),gic(10),gic(24),gic(12),gic(13),gic(14),1);
case ’g17’ % Change an element of an Ephemeris matrix
gicf_c(gic(2),gic(16),gic(25),gic(17),gic(18),gic(19),1);
% — Y Column Changes
case ’g7’ % Change an element of an Ephemeris matrix
gicf_c(gic(2),gic(4),gic(23),gic(6),gic(7),gic(8),2);
case ’g13’ % Change an element of an Ephemeris matrix
gicf_c(gic(2),gic(10),gic(24),gic(12),gic(13),gic(14),2);
case ’g18’ % Change an element of an Ephemeris matrix
gicf_c(gic(2),gic(16),gic(25),gic(17),gic(18),gic(19),2);
% — Z Column Changes
case ’g8’ % Change an element of an Ephemeris matrix
gicf_c(gic(2),gic(4),gic(23),gic(6),gic(7),gic(8),3);
case ’g14’ % Change an element of an Ephemeris matrix
gicf_c(gic(2),gic(10),gic(24),gic(12),gic(13),gic(14),3);
case ’g19’ % Change an element of an Ephemeris matrix
gicf_c(gic(2),gic(16),gic(25),gic(17),gic(18),gic(19),3);
% — NEXT Generator
case ’g9’ % Go to next row in the matrix and fill with zeros
gicf_d(gic(2),gic(4),gic(23),gic(6),gic(7),gic(8));
case ’g15’ % Go to next row in the matrix and fill with zeros
gicf_d(gic(2),gic(10),gic(24),gic(12),gic(13),gic(14));
case ’g20’ % Go to next row in the matrix and fill with zeros
gicf_d(gic(2),gic(16),gic(25),gic(17),gic(18),gic(19));
case ’g21’
etoud(gic(21)); % Put Contents of Editbox in UserData
case ’g22’
etoud(gic(22)); % Put Contents of Editbox in UserData
case ’savetag’
[agent] = isat3; % Initialize a satellite constellation
oldpath = cd;
[file_name, newpath] = uiputfile(’*_i.mat’, ’Save As: Initial Configuration File’);
if (ischar(file_name)==1)&(ischar(newpath)==1)
```

```
fprintf('Your current path is : %s\n ',newpath)
% Put an '_i' extension on the file if it doesn't already exist
if isempty(findstr(file_name,'_i'))
l = length(file_name);
file_name = [file_name(1:(l-4)),'_i',file_name((l-3):l)];
end
if (length(file_name)>2)
eval(sprintf('cd %s',newpath)) % e.g. 'cd C:\MATLAB\configs'
eval(sprintf('save %s agent',file_name));
eval(sprintf('cd %s',oldpath)) % e.g. 'cd C\MATLAB\gui'
set(hui(23),'String',file_name); % Update the main screen filename_r box
fprintf("'agent" configuration structure was saved as :%s\n',file_name)
end
end
case 'edittag'
oldpath = cd;
[file_name, newpath] = uigetfile('*_i.mat', 'Open: Initial Configuration File', 300, 200)
if (ischar(file_name)==1)&(ischar(newpath)==1)
if (length(file_name)>2) % Won't try and load files that don't exist
agent = []; % Eliminate the old agent structure
set(hui(23),'String',file_name); % Update the main screen filename_r box
eval(sprintf('cd %s',newpath)); % e.g. 'cd C:\MATLAB\configs'
eval(sprintf('load %s',file_name)); % Load our new agent
fprintf('Your file was loaded from path : %s\n ',newpath)
eval(sprintf('cd %s',oldpath)); % e.g. 'cd C\MATLAB\gui'
fprintf('%s loaded and now resident in local memory as the global structure "agent"\n',file_name)
gicf_edit % Run the screen loading file.
end
end
case 'maintag'
[object,figure_handle] = gcbo;
close(figure_handle); % Close up GICF and head back to main
otherwise
['No Valid Function Called in Generate Initial Condition Function (gicf.m)']
end
```

B.23 function gicf_a(range,satnum,position,row,x1,x2,x3)

```
% function gicf_a
% INPUT VARIABLES
% satnum: a char array for the # satellites handle
% position: a char array for the position tag, e.g. 'rp'
% row: a char array for the row tag, e.g. 'rowp'
% x1-x3: a char array for the x data tags, e.g. 'p1','p2','p3'
N = get(satnum,'UserData'); % Get the # of satellites
P = range*(rand(N,3)-.5); % Make a random assortment
set(position,'UserData',P); % Put P into rp's UserData
index = get(row,'UserData'); % Get rowp's UserData
set(x1,'String',num2str(P(index,1)));
set(x2,'String',num2str(P(index,2)));
set(x3,'String',num2str(P(index,3)));
```

B.24 function gicf_b(satnum,position,row,x1,x2,x3)

```
% function gicf_a
% INPUT VARIABLES
% satnum: a char array for the # satellites handle
% position: a char array for the position tag, e.g. 'rp'
% row: a char array for the row tag, e.g. 'rowp'
% x1-x3: a char array for the x data tags, e.g. 'p1','p2','p3'
P = get(position,'UserData'); % Get the position matrix
new_index = (get(row,'UserData')-1); % Get the current row index
if (new_index == 0), new_index = 1; end
vartoeud(new_index, row); % Puts new_index in rowp's
% USERDATA and STRING
set(x1,'String',num2str(P(new_index,1)));
set(x2,'String',num2str(P(new_index,2)));
set(x3,'String',num2str(P(new_index,3)));
```

B.25 function gicf_c(satnum,position,row,x1,x2,x3,col_index)

```
% function gicf_a
% INPUT VARIABLES
% satnum: a char array for the # satellites handle
% position: a char array for the position tag, e.g. 'rp'
% row: a char array for the row tag, e.g. 'rowp'
% x1-x3: a char array for the x data tags, e.g. 'p1','p2','p3'
% col_index: selects the row to change
P = get(position,'UserData'); % Get the current position matrix
index = get(row,'UserData'); % Current row index
if isempty(index), index = 1; end
column = ['x',num2str(col_index)]; % Pick one of the 3 rows (x1,x2,or x3)
if (col_index==1)
P(index,col_index) = str2num(get(x1,'String'));
elseif (col_index==2)
P(index,col_index) = str2num(get(x2,'String'));
else (col_index==3)
P(index,col_index) = str2num(get(x3,'String'));
end
set(position,'UserData',P) % Put P back in it's place
```

B.26 function gicf_d(satnum,ephemeris,row,x1,x2,x3)

```
% function gicf_d(satnum,ephemeris,row,x1,x2,x3)
% INPUT VARIABLES
% satnum: a char array for the # satellites handle
% ephemeris: a char array for the ephemeris tag, e.g. (P,T, or R)
% row: a char array for the row tag, e.g. 'rowp'
% x1-x3: a char array for the x data tags, e.g. 'p1','p2','p3'
P = udtovar(ephemeris); % Get the ephemeris matrix (P,T,or R)
sp = size(P,1); % length of P
N = udtovar(satnum); % Get the # of satellites
index = udtovar(row); % index points to a row in P
if (N > sp) % Fix any dimension problems
N = sp; % up front
vartoeud(N,satnum); % Update GUI screen
end
if (index ==0) % A bad condition
['Row index pointed to 0, so it was changed to 1']
new_index = 1;
P = [0,0,0];
elseif (index == N) % Pad with zeros
new_index = index + 1;
P = [P;[0,0,0]]; % Extend the matrix one more
elseif (index > N) % Go back to row 1
['Row index too large in gicf_d, so it was changed to 1']
new_index = 1; % Makes it go back
else
new_index = (index + 1); % Look at next row
end
if (new_index > N)
vartoeud(new_index,satnum) % Update the screen with a new satnum
end
vud(P,ephemeris); % Put P back after changing it
vartoeud(new_index,row); % Update the row counters
% Update the 3 Rows of P on the GICFGUI
set(x1,'String',num2str(P(new_index,1)));
set(x2,'String',num2str(P(new_index,2)));
set(x3,'String',num2str(P(new_index,3)));
```

B.27 function gicf_edit

```
% function gicf_edit
% Purpose: Take the existing 'agent (global)' structure in the matlab environment
% and load the variables into the UserData and Strings of the GICFGUI for Editing
% puposes. Once 'agent' is loaded into the GICFGUI UIcontrols, then you can fiddle with
% them to your hearts content and then save the results (under the same filename
% if you want).
% — Declare all GICF handles as global so that this function will see them —
global agent gic
% — Extract Defaults from AGENT —
if ~isempty(agent) % is the agent structure full?
vartoeud(agent.disp.color,gic(1)); % color box EUD
N = size(agent.phys,2); % number of satellites
vartoeud(N,gic(2));
vartoeud(agent.disp.dbit,gic(3)); % display bit
vP = []; vT = []; vR = []; % clear out existing matrices
for i = 1:N % Satellite ID Index
```

```
P = agent.phys(i).p; % Position Matrix
T = agent.phys(i).t; % Trajectory Matrix
R = agent.phys(i).r; % Rotation Moment Matrix
vP = [vP;P]; vT = [vT;T]; vR = [vR;R];
end
P = agent.phys(1).p; T = agent.phys(1).t; R = agent.phys(1).r;
vartoeud(agent.phys(1).m,gic(21));
vartoeud(agent.phys(1).i,gic(22));
for j = 1:3
vartoeud(P(1,j),gic(j + 5));
vartoeud(T(1,j),gic(j + 11));
vartoeud(R(1,j),gic(j + 16));
end
vud(vP,gic(4)); vud(vT,gic(10)); vud(vR,gic(16));
ve(1,gic(23)); ve(1,gic(24)); ve(1,gic(25));
else
['Agent was empty, so nothing was loaded in to GICFGUI'];
end % end of ~isempty routine
```

B.28 function gicf_init

```
% Generate Initial Conditions File Handle Initialization File
% —- gicf_init.m
% Finds the handles to all ui controls in the GICF GUI
% and initializes all variables upon startup.
% This script is called by the GICF callback on main
% ALL HANDLES ARE DECLARED GLOBAL
global agent gic
% Gather all handles and place them in a g() matrix
for i = 1:26
gic(i) = findobj('Tag',['g',num2str(i)]);
end
if ~isempty(agent)
gicf_edit; % Load current configuration into editor
else % Load default configuration into editor
% — Variable Definitions —
% AGENT.DISPLAY
vcolor = [0,0,1]; % Blue
vdbit = 0; % Low Impact Render
% AGENT.PHYSICAL
vsatnum = 3; % Number of Agents
vP = [0 0 0; 2 2 1; 3 0 -1]; % Position of 3 agents
vT = [-1 -1 -1; 5 5 5; -3 1 2]; % Trajectory Vectors
vR = vT + 1; % Rotational Moments
vmass = 10; % in Kilograms
vrinertia = 21.34; % Rotational Ineria of Sat
% — EDIT BOXES —
vartoeud(vcolor,gic(1)); % color editbox
vartoeud(vsatnum,gic(2)); % # Satellites
vartoeud(vdbit,gic(3)); % display bit default
vartoeud(vmass,g(21)); % mass of object
vartoeud(vrinertia,g(22)); % inertia of object
for i = 1:3 % Three column data
vartoeud(vP(1,i),gic(i+5)); % Position information
vartoeud(vT(1,i),gic(i+11)); % Trajectory information
vartoeud(vR(1,i),gic(i+16)); % Rotation information
vartoeud(1,g(i+22)); % Initial row index
end
% — PUSHBUTTONS —
vud(vP,gic(4));
vud(vT,gic(10));
vud(vR,gic(16));
end
```

B.29 function gicfgui()

```
% This is the machine-generated representation of a MATLAB object
% and its children. Note that handle values may change when these
% objects are re-created. This may cause problems with some callbacks.
% The command syntax may be supported in the future, but is currently
% incomplete and subject to change.
%
% To re-open this system, just type the name of the m-file at the MATLAB
% prompt. The M-file and its assocaitated MAT-file must be on your path.
load gicfgui
a = figure('Color',[0 0.4 0.6], ...
'Colormap',mat0, ...
'MenuBar','none', ...
'Name',' Initial Conditions File Editor (GICF)', ...
'NumberTitle','off', ...
'PointerShapeCData',mat1, ...
'Position',[14 108 700 600], ...
'Tag','gicfgui');
b = uicontrol('Parent',a, ...
'Units','points', ...
'BackgroundColor',[0 0 0.7], ...
'FontName','Arial', ...
'FontSize',9, ...
'FontWeight','bold', ...
etc... < see CD or email Dan Petrovich >
```

B.30 function hit_lights(action)

```
% Purpose: to turn on two lights from antipodal angles
switch(action)
case 1
lighting flat
light('Position',[300 300 300],'Style','infinite','Color',[1 1 1])
light('Position',[-300 -300 -300],'Style','infinite','Color',[.5 .5 .5])
material dull
```

```
case 0
lighting none
end
```

B.31 function [RP] = sense(N,SS,XP,NT,D,k_xmit,k_rec)

```
% [Received_Power] = sense(N,SS,XP,NT,D,k_xmit,k_rec)
% INPUT VARIABLES:
% agent_ID: Out of N agents, the one receiving incident_power passes agent_ID
% k_xmit: Transmitter, transmit efficiency (0-1)
% k_rec: Receiver, reception efficiency (0-1)
% OUTPUT VARIABLES:
% incident_power: This is a (6 x 1) array that updates the light intensity
% an agents orthogonal and antipodal receptors receive
% while passing through the cluster
RP = XP; % Quickly fills the array to the proper size
for agent_ID = 1:N % Over every satellite
RANGE = [1:(agent_ID-1) (agent_ID+1):N];
for k = 1:6 % Over each of your recievers
NET_POWER = 0; % Clear the power adder
n2 = SS(k,:,agent_ID); % Our sensor vector
for i = RANGE % Excluding you of course
nt=NT(:,agent_ID,i)'; % Normal vector pointing toward other sat.
dt2 = dot(nt,n2); % Components of receiver along 'nt'
if (dt2 > 0) % Non-Shadow condition
d = D(agent_ID,i); % Distance to the other guy
for j = 1:6 % Over Emitters faces
n1 = SS(j,:,i); % Emitter sensor vector
dt1 = dot(-nt,n1); % Component of emitter along 'nt'
if (dt1 > 0) % Non_Shadow condition
m1 = XP(i,j); % Emitter power
NET_POWER = NET_POWER + rint(d,m1,dt1,dt2,k_xmit,k_rec);
end
end
end
end
RP(agent_ID,k) = NET_POWER; % Received Power Matrix
end
end
```

B.32 function [agent] = isat3

```
% The following handles are used to access data in the gicfgui
% All refer to editable ephemerides.
global gic filename
% — Default Conditions Generator —
% A default configuration for the AGENT structure is defined
% ( ) implies an abreviation for the structure name
% # implies a user defined variable. The rest are defaults
% Major Class: AGENT (agent)
% 1. display (disp)
%# a) color (1 x 3) :a basic printable display color
% b) vertices (8 x 3)(vrt):the corners of a cube
% c) tvert (6 x 3) :the sensor vectors
% d) imap (K x K) :image map maps onto faces for great visuals
% g) cmap (K x 3) :colormap for imap
%# e) dbit (1 x 1) :decides display mode
% f) fac (6 x 4) :corner connect matrix
% g) xpower (xp) (scalar) :universal transmit power value (Watts)
% 2. physical (phys)
%# a) position (1 x 3)(p) :where the agent is located
%# b) trajectory (1 x 3)(t) :where the agent is going
%# c) rotation (1 x 3)(r) :how the agent is rotating
%# d) mass (1 x 1)(m) :how massive the agent is
%# e) Rinertia (1 x 1)(i) :the rotational inertia of an agent
% f) body (8 x 3)(b) : a rotated version of 'vert'
% g) tvector (6 x 3)(s) : a rotated version of 'tvect'
% h) transducers (trans)
% 0) rpower0 (6 x 1)(rp0): the incident power from channel 0
% 1) rpower1 (6 x 1)(rp1): the incident power from channel 1
% 2) rpower2 (6 x 1)(rp2): the incident power from channel 2
% 3) ecm (6 x 1) : emitter channel matrix {+X,-X,+Y,-Y,+Z,-Z}
% ****************************************************************
% — GET THE USER DEFINED VARIABLES FROM GICF USERDATA(s) —
% ****************************************************************
vcolor = [0 0 1]; % Default Blue for now
vdbit = udtovar(gic(3)); % Display bit
vxpower = 100; % Transmit Power (Watts)
if (vdbit<0)|(vdbit>2), vdbit = 0; end % Default to low-res graphics
N = udtovar(gic(2)); % Get the # of satellites
P = udtovar(gic(4)); % Get the Position Matrix
T = udtovar(gic(10)); % Get the Trajectory Matrix
R = udtovar(gic(16)); % Get the Rotation Moment Matrix
% — Clip the matrices off to the same length and re-save
MINN = min([N,size(P,1),size(T,1),size(R,1)]);
N = MINN(1); % Use only one smallest value
P = P(1:N,:); T = T(1:N,:); R = R(1:N,:);
vartoeud(N,gic(2)); % Update # Sats GUI
vmass = udtovar(gic(21)); % Get the mass
vrinertia = udtovar(gic(22)); % Get the Rotational Inertia
vfilename = udtovar(filename); % Get the filename
% ****************************************************************
% — ASSIGN THE DEFAULT VARIABLES —
% ****************************************************************
% — Enumerate the 8 vertices of a hexahedron
vert=[0 0 0;...
0 1 0;...
1 1 0;...
1 0 0;...
0 0 1;...
0 1 1;...
```

```
1 1 1;...
1 0 1];
vert = (vert - 0.5);
% — Enumerate the 6 sensors normal to 6 faces
tvert=[ 1 0 0;...
-1 0 0;...
0 1 0;...
0 -1 0;...
0 0 1;...
0 0 -1];
% — Connect each vertice using the 'fac' matrix
fac = [1 2 3 4;2 6 7 3;4 3 7 8; 1 5 8 4;1 2 6 5;5 6 7 8];
% ——— LOAD FACE IMAGE ———-
%[X,map] = imread('face1','bmp');
% **********************************************************************
% BEGIN FILLING THE STRUCTURE ARRAY
% **********************************************************************
% — Define the display data structure
display = struct('color',vcolor,...
'vert',vert,...
'tvert',tvert,...
'imap',zeros(10,10),...
'cmap',[],...
'dbit',vdbit,...
'fac',fac,...
'xp',vxpower);
% — Fill the physical structure with realistic default values
for i = 1:N % three satellites
RP0 = zeros(1,6); % Received Power on Channel 0
RP1 = RP0; % Received Power on Channel 1
RP2 = RP0; % Received Power Channel 2
ECM = zeros(1,6); % All emitters are orginally transmitting on
% the swarming channel (0)
stm = 0; % All satellites are originally floaters
% — Define the transducer data structure
transducer = struct('rp0',RP0,'rp1',RP1,'rp2',RP2,'ecm',ECM);
% — Define the physical data structure
physical(i) = struct('p',P(i,:),...
't',T(i,:),...
'r',R(i,:),...
'm',vmass,...
'i',vrinertia,...
'b',vert,... % Initially square with Ref-Frame
's',tvert,... % Initially square with Ref-Frame
'trans',transducer,...
'stm',stm);
end
%— Define the mother data class, the agent (a)
agent = struct('phys',physical,'disp',display);
```

B.33 *function [NBRS,flag] = local_sense(agent_ID,np)*

```
% Purpose: to look at the existing structure and determine if there
% are adjacency issues. If so, then it simply returns a flag of 0. If
% not, then the flat is set to 1 and local_sense.m
% returns a (6 x 1) neighbors matrix with neighbor IDs.
%
% INPUT VARIABLES
% NMAT: (6 x 3) list of antipodal positions from np
% STR: (60 x 60 x 60) structure storage matrix
% np: (1 x 3) (integer matrix)
% agent_ID: (scalar) (the index of the requesting satellite)
%
% OUTPUT VARIABLES:
% NBRS: (1 x 6) binary neighbors matrix
% flag: (scalar) binary 1 if sticking is allowed. 0 if not.
global NMAT
NBRS = zeros(1,6); % Default (1 x 6) zeros
% — Search for Local Neighbors in the +X,-X,+Y... etc positions
for q = 1:6;
NBRS(1,q) = get_element(NMAT(q,:) + np);
end
[bad_guy] = get_element(np); % Whomever is in your position. Better be nobody
there!
% If there are two numbers greater than zero in this position
if (bad_guy > 0);
flag = 0; % 0 implies that you are not allowed in that position
%sprintf('local_sense.m determined that position [%d,%d,%d]',np(1),np(2),np(3))
%fprintf('is already taken by satellite %1.0f.\n',bad_guy)
%fprintf('the connection was refused during this cycle.\n')
else
fprintf('local_sense.m detected %1.0f neighbors\n ',length(find(NBRS > 0)))
flag = 1;
end
```

B.34 *function main(action)*

```
global agent runn hui imap cmap color
main_back = findobj('Tag','main_back');
main_screen = findobj('Tag','main_screen');
if isempty(main_back)|isempty(main_screen)
['Empty main_back or main_screen handle encountered!!']
main_back
main_screen
end
switch(action)
case 'h15' % Lower range edit box
M = size(runn.f,2); % Number of available frames
range_check(hui(15),1,M,M); % keeps us honest
etoud(hui(15)); % put solution in editbox
case 'h16'
```

```
M = size(runn.f,2); % Number of available frames
range_check(hui(16),1,M,M); % keeps us honest
etoud(hui(16)); % put solution in editbox
case 'h19' % Satellite Image Rendering action
value = get(hui(19),'Value'); % Get the render condition from listbox
f = 'bsmall.bmp'; % default filename
color = [0 0 1]; % default blue render
d_bit = 1; % default to good rendering
switch(value)
case 1
d_bit = 0; % blue render
case 2
color = [.8 .8 .8]; % gray cube render
d_bit = 0;
case 3
f = 'bsmall.bmp'; % low-res b/w render
case 4
f = 'csmall.bmp'; % high-res b/w render
case 5
f = 'bbig.bmp'; % low-res color render
case 6
f = 'cbig.bmp'; % high-res color render
otherwise
f = 'bsmall.bmp'; % blue condition
d_bit = 0; % default to flat color rendering
end
% Image reading commands
wd1 = cd; cd ..;
wd2 = cd; cd([wd2,'\panels']);
[imap,cmap] = imread(f);
cd(wd1);
agent.disp.color = color; % set above
agent.disp.imap = imap; % (K x K) face image map if dbit = 1
agent.disp.cmap = cmap; % (K x 3) face color map if dbit = 1
agent.disp.dbit = d_bit; % set above
case 'h20' % Background slider
value = get(hui(20),'Value'); % Get the render condition from listbox
switch(value)
case 1
axes(main_back);
cla
set(main_back,'Color',[0 0 0]); % BLACK
axes(main_screen);
case {2,3}
axes(main_back);
cla
set(main_back,'Color',[1 1 1]); % WHITE
axes(main_screen);
case 4
f = 'mainback1.jpg'; % EARTH
case 5
f = 'mainback2.jpg'; % NEPTUNE
```

```
case 6
f = 'mainback3.jpg'; % PLUTO
case 7
f = 'mimas.jpg'; % MIMAS
otherwise
f = 'mainback1.jpg'; % EARTH
end
if ~((value==1)|(value==2)|(value==3))
% File reading function
wd1 = cd; cd ..;
wd2 = cd; cd([wd2,'\backgrounds']);
[imap,cmap] = imread(f); cd(wd1);
% Background Rendering commands
axes(main_back); % Background to the current axes
image(imap); % Splash the image – this erases the tag!!
set(main_back,'Tag','main_back'); % Cause image erases the Tag
axes(main_screen); % Foreground to the main axes
end
end
```

B.35 function main_gui()

```
% This is the machine-generated representation of a MATLAB object
% and its children. Note that handle values may change when these
% objects are re-created. This may cause problems with some callbacks.
% The command syntax may be supported in the future, but is currently
% incomplete and subject to change.
%
% To re-open this system, just type the name of the m-file at the MATLAB
% prompt. The M-file and its associtated MAT-file must be on your path.
load main_gui
a = figure('Color',[0.8 0.8 0.8], ...
'Colormap',[], ...
'MenuBar','none', ...
'Name','Modeling the Collective Behavior of Micro-Satellite Clusters', ...
'NumberTitle','off', ...
'Pointer','crosshair', ...
'PointerShapeCData',mat0, ...
'Position',[9 36 1000 660], ...
'Tag','main_figure');
b = uicontrol('Parent',a, ...
'Callback','activate', ...
'FontName','Arial', ...
'FontWeight','bold', ...
'Position',[700 135 150 19], ...
'String','RENDER FORMATION', ...
'Tag','run_main');
b = uicontrol('Parent',a, ...
'Callback','ric', ...
'FontName','Arial', ...
'FontWeight','bold', ...
```

```
’Position’,[550 135 150 19], ...
’String’,’RENDER IC’, ...
’Tag’,’ric_main’);
b = uicontrol(’Parent’,a, ...
’Callback’,mat1, ...
etc... <see CD or email Dan Petrovich >
```

B.36 function main_init

```
% Purpose: Generate an Initial Conditions File
%
% Every variable stored in the ’agent’ structure must be created to run a simulation
% This script will respond to the GICF button on the main gui
% In this GUI we can
% 1. EDIT an existing set of run parameters
% 2. Create a NEW run configuration
% 3. Save any results to a variables file of type (i_<filename>.mat)
%
global agent hui rui lights SP STR NMAT move O3 O8 SBODY SSENS
% — SPEED UP VARIABLES THAT NEVER CHANGE —
SP = []; % (NS x 3)A list of structural members positions
[NMAT] = make_nmat; % (27 x 3) local neighbors matrix
[move] = make_move(1); % movement matrix
O3 = ones(1,3); % (1 x 3) Matrix row repetition
O8 = ones(8,1); % (8 x 1) Matrix column repetition
% — LOAD DEFAULT AGENT STRUCTURE —
file_name = ’foursats_i.mat’;
eval(sprintf(’load %s’,file_name));
SBODY = agent.phys(1).b; % (8 x 3) Square body
SSENS = agent.phys(1).s; % (6 x 3) Square sensors matrix
% — Initial EditBox and TextBox Values —
TXT = [2:6,11:26];
vinit = {’00.00 sec’,’FORMATION’,’BEHAVIOR’,’# FRAMES’,’dt’,’10’,’0.20’,...
’LOWER F’,’UPPER F’,’1’,’10’,”,”,...
’blue|grey|bsmall|csmall|bbig|cbig’,...
’black|white|starry|earth|neptune|pluto|mimas’,...
’1:10’,’dt:0.20’,file_name,’form3a_r’,’behavior1_b’,’et:00.00’};
rinit = {’>’,’I’,’R’,’R’,’RENDER FORMATION OPTIONS’};
% — Find all of the handles for the data entry uicontrols —
for i = 1:21
hui(TXT(i)) = ehandles([’h’,num2str(TXT(i))],vinit{i});
end
for j = [1,7,8,9,10],
hui(j) = findobj(’Tag’,[’h’,num2str(j)]);
end
% — Find all handles for 3 light box —
for i = [1,2,3,4,5]
rui(i) = findobj(’Tag’,[’rr’,num2str(i)]); % get the handles
set(rui(i),’String’,rinit{i}); % Set rinit values
end
vud(1,rui(1)); % Make the Activate_I render the default
```

```
% — Initialize the data in each of the data entry uicontrols —
set(hui(1),’Max’,str2num(vinit{6}),’Min’,1);
set(hui(1),’Value’,1);
set(hui(11),’Callback’,’etoud(gcbo);’);
set(hui(12),’Callback’,’etoud(gcbo);’);
set(hui(15),’Callback’,’main(”h15”);’);
set(hui(16),’Callback’,’main(”h16”);’);
set(hui(19),’Callback’,’main(”h19”);’);
set(hui(19),’Value’,1);
set(hui(20),’Callback’,’main(”h20”);’);
set(hui(20),’Value’,1);
lights = findobj(’Tag’,’lights’);
set(lights,’Value’,0);
```

B.37 function [mood0] = make_mood0(mass,xpower)

```
% Purpose: to put behavior pertinent variables in a structure called mood0
% for easy portability.
% Mood 0 controls swarming activity
% Mood 1 controls attraction
% Mood 2 turns of transmiiters
% THIS SETUP WORKED PREVIOUSLY
%cs = 1.50; % Meters Safe collision radius
%de = 2*cs; % Meters Equilibrium distance (radius)
%abar = -1.0; % meters/second Maximum attractive field force
%rbar = 0.2; % meters/second Maximum repusive field force
%imax = (2.679/mass); % meters/second Thruster impulse divided by mass
%mood0 = struct(’xp’,xpower,’cs’,cs,’de’,de,’a’,abar,’r’,rbar,’imax’,imax);
P = xpower;
del = 5; % Equilibrium distances in meters
k0 = real(sqrt(P/(4*pi*del^2))); % Equilibrium power
A = 0.9; % Pre-multiplier. Like an amplitude term
mood0 = struct(’k’,k0,’A’,A);
```

B.38 function [mood1] = make_mood1(mass,radius,xpower)

```
% Purpose: to put behavior pertinent variables in a structure called mood0
% for easy portability.
%cs = .1*radius; % Meters Safe collision radius
%de = .2*radius; % Meters Equilibrium distance (radius)
%abar = -1.0; % meters/second Maximum attractive field force
%rbar = 0.2; % meters/second Maximum repusive field force
%imax = (2.679/mass); % meters/second Thruster impulse divided by mass
%mood1 = struct(’xp’,xpower,’cs’,cs,’de’,de,’a’,abar,’r’,rbar,’imax’,imax);
P = xpower;
del = 0.2; % Equilibrium distances in meters
%k1 = P/(4*pi*del^2); % Equilibrium power
k1 = real(sqrt(P/(4*pi*del^2))); % Constant
A = 1.3; % Pre-Multiplier. Like an amplitude term
```

```
mood1 = struct('k',k1,'A',A);
```

B.39 function [move] = make_move(width)

```
move(1,:) = [width 0 0]; % +X
move(2,:) = -move(1,:); % -X
move(3,:) = [0 width 0]; % +Y
move(4,:) = -move(3,:); % -Y
move(5,:) = [0 0 width]; % +Z
move(6,:) = -move(5,:); % -Z
```

B.40 function [NMAT] = make_nmat

```
% Purpose: to make a set of positions about the point 0,0,0 at which to look
% for neighboring structural elements
%count = 0;
%for i = -1:1
% for j = -1:1
% for k = -1:1
% count = count + 1;
% NMAT(count,:) = [i j k];
% end
% end
%end
NMAT = [1 0 0;-1 0 0;0 1 0;0 -1 0;0 0 1;0 0 -1];
```

B.41 function [RT0] = make_RT0(N)

```
%*** INITIAL ROTATION MATRICES (NO ROTATION)(3 x 3) identity matrices
for x = 1:N, RT0(:,:,x) = eye(3,3); end
```

B.42 function [R] = mcad2(A,Rin,theta0,theta1)

```
% function [R] = mcad2(A,Rin,theta0,theta1)
% A does not have to be normalized in this function
if (theta0==theta1)
R = Rin;
else
Anorm = A;
x = Anorm(1); y = Anorm(2); z = Anorm(3);
c = cos(theta1); s = sin(theta1);
xs = x^2; ys = y^2; zs = z^2;
xy = x*y; xz = x*z; yz = y*z;
mc = (1-c);
alf= (ys+zs);
```

```
bs = (alf + xs);
b = sqrt(bs);
f23 = ((c*(xs-bs) + alf)*yz)/(alf*bs);
xzb = x*s/b; yzb = y*s/b; zzb = z*s/b;
R(1,1)=(alf*c + xs)/bs;
R(1,2)=(mc*xy/bs) + zzb;
R(1,3)=(mc*xz/bs) - yzb;
R(2,1)=(mc*xy/bs) - zzb;
R(2,2)=(xs*c + alf)*ys/(alf*bs) + zs*c/alf;
R(2,3)= xzb + f23;
R(3,1)=(mc*xz/bs) + yzb;
R(3,2)= f23 - xzb;
R(3,3)= (xs*c + alf)*zs/(alf*bs) + ys*c/alf;
end
```

B.43 function moveabsolute(handle,dist,az,el)

```
% Purpose: to take the camera to an absolute distance, azimuth, and elevation
% as specified on the input. Handle specifies the axis we you are looking at.
set(handle,'CameraTarget',[0 0 0]);
set(handle,'View',[az,el]); % Move to specific angle
cpos = get(handle,'CameraPosition'); % Returns the current camera position
nc = norm(cpos);
if (nc==0)
nc=1;
['Warning, norm of Camera Position is zero, so set to one.']
cpos
handle
end
% Cpos is relative to the origin
unit = cpos*(1/nc); % Unit vector towards camera
newcp = unit*dist; % A point distance 'dist' from ctarg
set(handle,'CameraPosition',newcp) % Changes the existing camera position
```

B.44 function movecamera(fraction)

```
%The movecamera function will zoom all the way in in fraction = 100
%It will zoom all the way out if fraction = -100
cpos = get(gca,'CameraPosition'); % Returns the current camera position
%ctarg = get(gca,'CameraTarget'); % Returns the current target position
ctarg = [0 0 0];
newcp = cpos - (fraction/100)*(cpos - ctarg);% Converts the zoom factor into a new
camera position
set(gca,'CameraPosition',newcp) % Changes the existing camera position
set(gca,'CameraTarget',[0 0 0]);
```

B.45 function [D, VOUT] = n_vector(VIN)

```
% function [Magnitude,UnitVector] = n_vector(VIN)
% O3 = (3 x 1)
% D = (N x 1)
% VOUT = (N x 3)
% VIN = (N x 3)
O3 = ones(3,1);
D = sqrt((VIN.*VIN)*O3);
VOUT = (VIN + eps)./((D + eps)*O3');
```

B.46 function [NT,P12,D] = near_sats(P)

```
%------------------------------------------------------------
%function [NT,P12,D] = near_sats(P)
%—- INPUT VARIABLES:
%P: the satellite position matrix (you, a satellite, and your buddies)
%—- OUTPUT VARIABLES:
%D: a distance matrix of dimension (length(P), 1)
% in which element ”YourIndex” is 0 (distance to yourself)
% History: This Code written en route to Dayton, OH from Evansville, IN
% In John's car on Pat's laptop.
% Shania Twain was incredible in concert! And makes for good code!
%------------------------------------------------------------
LP = size(P,1); % The number of satellites swarming including you
O = ones(1,LP); % Pre multiply matrix for repeating columns
% See Journal for visual description of this code
OP = O'; % Transpose
Px = P(:,1); % First column of P
Py = P(:,2); % Second column of P
Pz = P(:,3); % Third column of P
Dx = (OP*Px' - Px*O); % Dx (1 x N x N)
Dy = (OP*Py' - Py*O); % Dy (1 x N x N)
Dz = (OP*Pz' - Pz*O); % Dz (1 x N x N)
P12(1,:,:) = Dx;
P12(2,:,:) = Dy;
P12(3,:,:) = Dz;
D = sqrt(Dx.^2 + Dy.^2 + Dz.^2);
NT(1,:,:) = (Dx+eps)./(D+eps);
NT(2,:,:) = (Dy+eps)./(D+eps);
NT(3,:,:) = (Dz+eps)./(D+eps);
```

B.47 function overnight

```
global runn
% Purpose: to check the GUI overnight button and save the current runn structure
if
% it is depressed. The file is saved as overnight.mat unless you change it.
% — Check the Overnight Save button with Tag 'check3'
```

```
h = findobj('Tag','check3');
value = get(h,'Value');
if (value==1),
save overnight runn; % Save the runn structure in the file 'overnight'
fprintf('%s\n','File <overnight.mat> successfully saved in this directory');
end
```

B.48 *function [DS] = p_field(MW,UW,mood)*

```
global O3
% function [Ds] = p_field(Watts,Wnorm,mood)
% Purpose: Generate an attractive or repulsive "sensation"
% in the form of a "Desired Velocity" (Ds) that a satellite feels in the
% presence of other satellites.
%
% INPUT VARIABLES
% de: (scalar) zero force point. Equilibrium distance
% at this point
% mood0: (structure)
% OUTPUT VARIABLES
% DS: (scalar) postitive = repulsion (meters/second)
% negative = attraction (meters/second)
% Constraints:
% de ~= Cs
% a ~= 0
% each MW ~= 0
% RE-INSTATE THIS STUFF IF THE OTHER ATTEMPT DOESN'T WORK
%P = mood.xp;
%Cs = mood.cs;
%de = mood.de;
%a = mood.a;
%r = mood.r;
%d = sqrt(P./(4*pi*MW + eps));
%K1 = (de - d)/(de - Cs);
%K2 = (1 - r/a);
%MDS = real(a*(1-K2.^K1));
%DS = -UW.*(MDS*ones(1,3));
%for j = find(MW < 8e-07)'; % 1000 meters at 100 watts
% DS(j,:) = [0 0 0]; % Gets rid of zero problems
%end
% END RE-INSTATE
% Ds(find(Ds>r))=r; % Clipping may be counterproductive.
d = MW;
for j = find(MW < 8e-07)'; % 1000 meters at 100 watts
UW(j,:) = [0 0 0]; % Gets rid of zero problems
end
k = mood.k; % constant for response equation
A = mood.A; % amplitude term
MDS = A*((MW - k)./(MW + k)); % The whole swarming behavior lies herein!!
```

```
DS = -UW.*(MDS*O3);
```

B.49 function plot_sat(vertices,pos0,pos1)

```
global fac color imap cmap d_bit
% function plot_sat(vertices,pos0,pos1)
% INPUT_VARIABLES
% vertices : (8 x 3) standard body frame matrix
% fac : (6 x 4) standard vertice index matrix
% traj : (1 x 3) trajectory unit vector, direction of travel
% color : (1 x 3) RGB agent_color for surface agent_color
% imap : (K x K) satellite face
% cmap : (N x 3) agent_colormap associated with ”image”
% d_bit : (1 x 1) switches between high or low resolution rendering
% OUTPUT_VARIABLES
% — none —
% The only output is a cube defined by vertices and faces
%
% You should set ’axis vis3d’ and ’axis(’off’)’ before using this function
% To change the number of cases, you must also go to isat3.m and change the
% error correcting line for DBIT.
switch(d_bit)
case 0 % LOW RESOLUTION (FLAT COLOR) MODE
patch(’faces’,fac,’vertices’,vertices,’FaceColor’,color,’EdgeColor’,[.4 .4 .4]);
case 1 % HIGH RESOLUTION (RENDERED IMAGE) MODE
hold on
for n = 1:6 % One face for each side
for i = 1:3 % One for each dimension
S(:,:,i) = [vertices(fac(n,1),i) vertices(fac(n,2),i);...
vertices(fac(n,4),i) vertices(fac(n,3),i)];
end
s1 = surf(S(:,:,1),S(:,:,2),S(:,:,3));
% Describe a surface, then paint a texture on it
set(s1,’FaceColor’,’texturemap’,’EdgeColor’,’none’,’CData’,imap);
colormap(cmap);
% In this instance agent_colordata is a (K x K) matrix that looks like
% the face of a satellite.
end
hold off
case 2 % DOT TRACKING MODE
% It just so happens that vectors 1 and 7 of ’vects’ are antipodal, so...
p1 = pos0;
p2 = (0.5*pos1 + 0.5*pos0);
line([p1(1) p2(1)],[p1(2) p2(2)],[p1(3) p2(3)],’Color’,[.5 .5 .5]);
otherwise
fprintf(’No valid graphics case selected in plot_sat.m \n’)
end
```

B.50 function [P0,T0,R0,B0,S0,ECM,stm0] = pull(runnfm1);

```
P0 = runnfm1.P; % Position in the last frame
T0 = runnfm1.T; % Pass Tangential Velocity to Behavior later
R0 = runnfm1.R; % Pass Angular Velocities to Behavior later
B0 = runnfm1.B; % Body Vectors
S0 = runnfm1.S; % Sensor Vectors
ECM = runnfm1.ECM; % Emitter Channel Matrix
stm0 = runnfm1.stm; % Structure membership
```

B.51 function [dR,A] = rupdate(T,O3,dt)

```
Rm = sqrt(sum(T.^2,2)); % magnitude of motion in (m/s)
A = (T + eps)./(Rm*O3 + eps); % eps effectively thwartz zero movement conditions
dR = dt*Rm; % dTHETA matrix (N x 1) in (radians/sec)
```

B.52 function [P,A,theta] = randomP(N, satwidth, centersats)

```
%——————————————————————————————
%function [P,R] = randomP(N, satwidth, centersats)
%—- INPUT VARIABLES:
%N: the desired number of swarming satellites
%widthsats: the desired width of each hexahedronal satellite
%centersats: the desired center for the swarming cluster
%—- OUTPUT VARIABLES:
%P: position matrix of dimension (N x 3)
%A: vector of rotation matrix of dimension (N x 3)
%theta: rotation matrix in radians (N x 1)
%——————————————————————————————
variance = 10; % Any pre-multiplier will increase the
angle = 360;
A = (pi/180)*(angle*rand(N,3)); % Rotate the satellites some random amount
P = variance*randn(N,3); % Random selection of positions
D = [0];
[D] = near_sats(P); % Find the inter-satellite distance matrix
['First cluster created']
while (length(find(D<=satwidth))>N)
[D] = near_sats(P); % Returns a diagonal distance matrix
['Seeking a valid cluster']
end
theta = rand(N,1)*(2*pi);
```

B.53 function range_check(handle,lower,upper,default);

```
% function range_check(handle,lower,upper,default);
% Purpose: Check to see if the contents of HANDDLE are valid
% in the RANGE [LOWER to UPPER].
% If not, then range_check inserts the DEFAULT
```

```
%
Sin = get(handle,'String');
if (isempty(str2num(Sin)))
set(handle,'String',num2str(default));
else
S = str2num(Sin);
if (S<lower)|(S>upper)
set(handle,'String',num2str(default));
end
end
```

B.54 *function [BODY,SENS] = render_main(m,rt1,b0,s0,CG0,CG1)*

```
% Purpose: to rotate, translate, then render a single cube in the swarm
global O8
ROTATE = rt1; % (3 x 3) Rotation Vector
BODY = b0*ROTATE; % Rotate the body
SENS = s0*ROTATE; % Rotate the sensors
VP = (BODY + O8*CG1); % Translated Body
% RENDER ONE SATELLITE
plot_sat(VP,CG0,CG1);
```

B.55 *function ric*

```
% function ric
% Purpose: To Splash the initial swarm configuration on the main
% graph screen according to the settings in 'agent.disp'
% INPUT VARIABLES
% none
% OUTPUT VARIABLES
% none
% — splash to main screen —
% INTERNAL VARIABLES
% agent: type = structure, holds all ephemeris information
% < see program >
global agent fac color imap cmap d_bit O8
N = length(agent.phys); % The number of agents in the cluster
clear_main; % This script clears and resets the main screen
% — Load all defaults —
fac = agent.disp.fac; % (6 x 4)
color = agent.disp.color; % (1 x 3)
imap = agent.disp.imap; % (K x K)
cmap = agent.disp.cmap; % (K x 3)
d_bit = agent.disp.dbit; % scalar/binary
if (d_bit==2), display_bit = 0; end
pos0 = [0 0 0]; pos1 = [0 0 0];
% — Render each of the N agents in the constellation —
for i = 1:N
% Assign the body frame (8 x 3) to Vertices
```

```
V = agent.phys(i).b; % b (8 x 3) is for body
P = agent.phys(i).p; % p (1 x 3) is for body cg in space
VP = (V + O8*P); % offset body frame
plot_sat(VP,pos0,pos1);
end
hit_lights(1)
drawnow
fprintf('%s\n','Render Initial Configuration (RIC.M) invoked')
hold off
```

B.56 *function [m2] = rint(d,m1,dt1,dt2,k1,k2);*

```
% function [m2] = rint(m1,n1,n2,k1,k2,nt,d)
% INPUT VARIABLES
% m1: The transmit power of the surface emitter
% n1: Normalized vector normal to the transmission face (surface emitter)
% n2: A vector normal to the receiver face (surface light receiver)
% k1: The transmission coefficient of transmitter n1
% k2: The reception coefficient of receiver n2
% p12: A vector from the (RECEIVER -> EMITTER) (P2 - P1) (1 x 3)
% d: The distance from the receiver to the emitter
%
% OUTPUT VARIABLES
% m2: The magnitude of a light received from (n1)
%
% INTERNAL VARIABLES
% n1: The normal version of n1
% m1: The magnitude of n1
% nt: The normalized vector from P1 to P2 = (P2-P1) = 21
% NOTE:
% 1 implies EMITTER
% 2 implies RECEIVER (US)
numor = (m1 * k1 * k2 * dt1 * dt2); % Effective tranmit power with slant angel
denom = (4 * pi * d^2); % Denominator
% If you get a warning because of a divide by zero in m2, then it means that
% two satellites are in the same position. It is most probably because you had
% a satellite in the same position as a structural element originally and it
% did not move during swarming.
m2 = (numor / denom); % received power from emitter (1)
```

B.57 *function [RT1] = rt_update(dR0,dR1,A,RT0,N);*

```
% Rotate each body in space FOR THIS FRAME
for i = 1:N
theta0 = dR0(i,:); % Previous rotation angle
theta1 = dR1(i,:); % Current rotation angle
R0 = RT0(:,:,i); % Initially all identity matrices
A0 = A(i,:); % Rotation vector
% theta1 is the old rotation angle, theta2 is the new rotation angle
```

```
[R1] = mcad2(A0,R0,theta0,theta1);
RT1(:,:,i) = R1; % Save the new rotation matrix
end
```

B.58 function save_runn(action)

```
global runn hui
switch(action)
case ’save_runnfile’
if isempty(runn)
fprintf(’Your runn file is empty; therefore, saving is an invalid action.\n’);
break
end
oldpath = cd;
[file_name, newpath] = uiputfile(’*_r.mat’, ’Save As: Initial Configuration File’);
if (ischar(file_name)==1)&(ischar(newpath)==1)
fprintf(’Your current path is : %s\n ’,newpath)
% The ’_r’ extension suggests that this is a runn file
% Put an ’_r’ extension on the file if it doesn’t already exist
if isempty(findstr(file_name,’_r’))
l = length(file_name);
file_name = [file_name(1:(l-4)),’_r’,file_name((l-3):l)];
end
if (length(file_name)>2)
eval(sprintf(’cd %s’,newpath)) % e.g. ’cd C:\MATLAB\configs’
eval(sprintf(’save %s runn’,file_name));
eval(sprintf(’cd %s’,oldpath)) % e.g. ’cd C\MATLAB\gui’
set(hui(24),’String’,file_name); % Update the main screen filename_r box
fprintf(’Agent ”runn” configuration structure was saved as :%s\n’,file_name)
end
end
case ’load_runnfile’
oldpath = cd;
[file_name, newpath] = uigetfile(’*_r.mat’, ’Open: Initial Runn File’, 300, 200);
if (ischar(file_name)==1)&(ischar(newpath)==1)
if (length(file_name)>2) % Won’t try and load files that don’t exist
runn = []; % Eliminate the old runn structure
set(hui(24),’String’,file_name); % Update the main screen filename_r box
eval(sprintf(’cd %s’,newpath)); % e.g. ’cd C:\MATLAB\configs’
eval(sprintf(’load %s’,file_name)); % Load our new agent
fprintf(’Your file was loaded from path : %s\n ’,newpath)
eval(sprintf(’cd %s’,oldpath)); % e.g. ’cd C\MATLAB\gui’
fprintf(’%s loaded and now resident in local memory as the variable ”runn”\n’,file_name)
end
end
end
```

B.59 *function [RP0,RP1,RP2,A] = sense(N,SS,ECM,NT,*

```
D,stm,k_xmit,k_rec,xpower,radius)
% function [RP0,RP1,RP2,A] = sense(N,SS,ECM,NT,D,stm,k_xmit,k_rec,xpower,radius)
% INPUT VARIABLES:
% N: The number of satellites
% SS: (6 x 3 x N) Sensor vector matrix (all normal vectors)
% ECM: (N x 6) Transmit channel {0 1 2}
% xpower: (scalar) Universal Emitter Power (Watts)
% NT: (3 x N x N) Inter-Sat normal matrices
% D: (N x N) Inter-Sat distances, i.e. D(1,2) is dist(1->2)
% k_xmit: Transmitter, transmit efficiency (0-1)
% k_rec: Receiver, reception efficiency (0-1)
% OUTPUT VARIABLES:
% RP0: (N x 6) Received Power on Channel 0
% RP1: (N x 6) Received Power on Channel 1
% RP2: (N x 6) Received Power on Channel 2
% A: (N x 3) Sticking Assignment list {his ID, your ID, your face #}
% INTERNAL VARIABLES:
% agent_ID: Out of N agents, the one receiving incident_power passes agent_ID
AA = []; % Clear the assignments list
RP0 = zeros(N,6); % Clean slate
RP1 = RP0; % Clean slate
RP2 = RP0; % We never receive power on this channel
sq = 0;
m1 = xpower; % Uniform Transmit power
[CLIST] = find(stm==0); % Find all Cluster (swarming) satellite ID #s
% Clist only updates the received power for swarming elements!
% This results in a significant speed improvement near the end of construction
for agent_ID = [CLIST] % The set of all swarming (motive) satellites
RANGE = [1:(agent_ID-1) (agent_ID+1):N];
for k = 1:6 % Over each of your recievers
NET_PWR0 = 0; % Clear the power adder
NET_PWR1 = 0; % Clear the power adder
NET_PWR2 = 0; % Clear the power adder
n2 = SS(k,:,agent_ID); % Our sensor vector
for i = RANGE % The set of all satellites except you
nt = NT(:,agent_ID,i); % Normal vector pointing toward other sat.
dt2 = (n2*nt); % dot product of two unit vectors
if (dt2 > sq) % Non-Shadow condition
d = D(agent_ID,i); % Distance to the other satellite
for j = 1:6 % Over Emitters faces
n1 = SS(j,:,i); % Emitter sensor vector
dt1 = (-n1*nt); % dot product of two unit vectors
if (dt1 > sq) % Non_Shadow condition
%if ((d < radius)&(stm(i) < stm(agent_ID))&(ECM(agent_ID,k)==1));
% You are close enough, he is structural, and his face is active.
if (d<radius)&(stm(i)==1)&(ECM(i,j)==1)
AA = [AA;[agent_ID,i,j]]; % Sticky assignment matrix
end
channel = ECM(i,j); % Emitter channel matrix
switch(channel)
case 0
```

```
NET_PWR0 = NET_PWR0 + rint(d,m1,dt1,dt2,k_xmit,k_rec);
case 1
NET_PWR1 = NET_PWR1 + rint(d,m1,dt1,dt2,k_xmit,k_rec);
otherwise
%fprintf('No case reached in sense \n')
end
end
end
end
end
RP0(agent_ID,k) = NET_PWR0; % Received Power Matrix
RP1(agent_ID,k) = NET_PWR1; % Received Power Matrix
% RP2(agent_ID,k) = NET_PWR1; % Received Power Matrix (NOT USED)
end
end
% This routine cuts A down to size because it repeats 3 times per entry
A = [];
if ~isempty(AA) % Not empty
for e = 1:6:size(AA,1); % Thin out the A list by 1/3
A = [A;AA(e,:);AA(e+1,:)];
end
% [A] = afilter(A); % Make the A-list feesible in reality
end
% You are added to the stick list (A) if:
% 1. You are a swarming element, i.e. stm = 0 for you
% 2. You are within a distance 'radius' of a structural element, i.e. stm = 1
% 3. Your transmitting side is currently active, i.e. ECM(you,side) = 1
```

B.60 script sixpack.m

```
% This is a high level program that finds the received light
% on all six orthogonal and antipodal sensors mounted on a Hexahedronal Satellite
% Note: The sum of ever increasing numbers from 1 to n is n(n+1)/2
for N = 2:15 % This can go from 2 to 21 in 15-16 hours
[agent_old] = structure(N); % Load the agent structure with random numbers
agent_new = agent_old; % Fill the new agent structure full of random entries
k_xmit = 1; % The transmission loss coefficient (gain)
k_rec = 1; % The reception loss coefficient (gain)
S = 0; % Counting variable for # shadows
max = 200; % There are "max" random clusters created
tic % Reset the "Stopwatch"
for k = 1:max % Generate "max" random clusters and gather stats on them
[agent_old] = structure(N);% Re-generate a new random cluster
S = 0; % Clear the "Number of Shadows" variable
for agent_ID = 1:N % Find the incident power on each satellite
[incident_power] = incident(agent_old, agent_new, agent_ID, k_xmit, k_rec);
agent_new(agent_ID).transducers.received_power = incident_power';
S = S + length(find(incident_power==0));
end % Sum the number of shadows cast
Zeros(k) = S; % Store the total number of shadows per instance
end % of a random cluster
```

```
t_total = toc; % Keeps track of execution time
% — Display Internal Data —
['The average percentage of shadows was ',num2str(100*sum(Zeros)/(6*N*max))]
['The standard deviation over different random clusters was ',num2str(std(Zeros))]
['The total number of shadows found is ',num2str(sum(Zeros)), ' out of ',num2str(6*N*max),'
cast.']
['The average execution time for N satellites was ',num2str(t_total/max),' seconds']
ATIME(N) = t_total/max; % Average execution time per cluster
SDRC(N) = std(Zeros); % STD of shadows over "max" different clusters
APS(N) = 100*sum(Zeros)/(6*N*max); % Average Percentage of Shadows cast over a
total of
end % 'max' clusters of cardinality N
% — Plotting Routine —
plot([2:15],APS(2:15))
set(gcf,'Color',[1,1,1])
xlabel('The Number of Satellites in a Random Cluster')
ylabel('The Percentage of Completely Shadowed Faces')
title('The Number of Hexahedronal Satellites vs. Percentage of Shadowed Faces')
gtext({['Xmit Gain: ',num2str(k_xmit)],...
['Rec. Gain: ',num2str(k_rec)],...
['Total Computation Time: ',num2str(16),' hours'],...
['Number of Clusters Analyzed: 46,000']})
% — Write the Plot to 'filename.bmp' —
[X,map] = capture(1);
imwrite(X,map,'sixpack2.bmp','bmp');
save vars_6pack
```

B.61 Structural Emergence Simulator Modeling Code (STEMS) MAIN

```
% Structural Emergence Simulator
% ver. 1.01
close all
clear agent
% Run the main GUI
maingui
% Initialize the Environment
main_init
% Initialize the Viewer
viewer_init
```

B.62 function [A] = stick_list(D,stm,radius)

```
% Stick_List
% Given D, ECM, stm
[i,j] = find(D < radius);
E = [i,j];
U = [];
for k = 1:length(E)
if (E(k,1)< E(k,2))
```

```
U = [U;E(k,:)];
end
end
U = sortrows(U);
J = [];
for k = 1:length(U)
A1 = stm(U(k,1));
A2 = stm(U(k,2));
if (A1<A2)
J = [J;[U(k,2),U(k,1)]];
elseif (A1>A2)
J = [J;U(k,:)];
end
end
A=[];
if (length(J)>0)
A = J(1,:);
PL = J(1,2);
for k = 2:length(J)
if (length(find(PL==J(k,2)))==0)
A = [A; J(k,:)];
PL = [PL J(k,2)];
end
end
end
```

B.63 function [agent] = structure(N);

```
% This Program is used to play with structure organization for the Clustered Satel-
lite
% Simulator ver 1.0 (CSS 1.0)
for i = 1:N
Pin = 10*(rand(1,3)-.5); Rin = (rand(1,3)-.5);
Tin = (rand(1,3)-.5); Bin = (rand(8,3)-.5);
RP = zeros(6,1); TP = 100*rand(6,1);
for n = 1:6
N_vect = (rand(1,3)-.5); No_vect = norm(N_vect);
Mx(n,:) = N_vect/No_vect;
end
agent(i) = struct(’membership’,0,’position’,Pin,’rotation’,Rin,...
’trajectory’,Tin,’body’,Bin,...
’transducers’,struct(’received_power’,RP,’transmit_power’,...
TP,’matrix’,Mx));
end
%agent. 1.438 kilobytes per field
%1xN struct array with fields:
% membership
% position
% rotation
% trajectory
% body
% transducers.received_power
% transducers.transmit_power
% transducers.matrix
```

B.64 function [dTn] = tupdate(T,O3,dt)

```
Tm = sqrt(sum(T.^2,2))*O3; % magnitude of motion in (m/s)
Tn = (T + eps)./(Tm + eps); % eps effectively thwartz zero movement conditions
dTm = (dt*Tm); % change in position
dTn = dTm.*Tn; % vectorized change in position
```

B.65 function udtoe(from_tag,to_tag)

```
% function etoud(localtag)
% Purpose: to take data out of somebody's UserData and put
% it in someone elses userdata and edit box
data = get(from_tag,'UserData');
set(to_tag,'String',data);
set(to_tag,'UserData',data);
```

B.66 function [variable] = udtovar(handle);

```
% function [variable] = udtovar(handle);
% Purpose: Put what's in handle's UserData into a Variable
variable = get(handle,'UserData');
```

B.67 function [ECM] = update_ecm(ECM,NBRS,np,ce)

```
% Purpose: after sticking to the structure, we need to update the active
% faces and turn off old transmitters. This routine effecively does that
i = find(NBRS > 0); % All existing neighbors
ecm1 = ECM(ce,:); % This is your channel matrix
for q = 1:length(i);
ecm0 = ECM(NBRS(i(q)),:); % Emitter Channel Matrix for agent_ID = NBRS(q)
switch i(q) % The side index
case 1 % The +X case
ecm0(2)=3; % Turn off the other guys -X side
ecm1(1)=3; % Turn off my side
case 2 % The -X case
ecm0(1)=3; % Turn off it's +X side
ecm1(2)=3; % Turn off my side
case 3 % The +Y case
ecm0(4)=3; % Turn off it's -Y side
ecm1(3)=3; % Turn off my side
case 4 % The -Y case
```

```
ecm0(3)=3; % Turn off it's +Y side
ecm1(4)=3; % Turn off my side
case 5 % The +Z case
ecm0(6)=3; % Turn off it's -Z side
ecm1(5)=3; % Turn off my side
case 6 % The -Z case
ecm0(5)=3; % Turn off it's +Z side
ecm1(6)=3; % Turn off my side
end
ECM(NBRS(i(q)),:) = ecm0; % Update the ECM matrix
end
% MODIFY YOUR ACTIVE FACES BASED UPON SOME USER ENTERED FUNCTION f(x,y)
x0 = np(1); y0 = np(2);
x1 = (x0 - 0.5); y1 = (y0 - 0.5);
x2 = (x0 + 0.5); y2 = (y0 + 0.5);
X = [x1;x1;x2;x2]; Y = [y1;y2;y1;y2];
% ******************************************************************
Z = f(X,Y); % THIS IS THE KEY TO BEHAVIOR !!!!! F(X,Y)
% Z is a (4 x 1) matrix of Z values that describe a small surface
% whos corners correspond to those of the current agent in X and Y
% ******************************************************************
NZ = round(Z - np(3)); % Centered and rounded to the nearest integer
nz1 = NZ(1); nz2 = NZ(2);
nz3 = NZ(3); nz4 = NZ(4);
j = find(NBRS==0); % Possible sides that we can activate
for q = 1:length(j);
switch j(q) % The side index
case 1 % +X case
if (nz3==0)&(nz4==0) % +X side
ecm1(1)=1; % Activate!
end
case 2 % -X side of cube
if (nz1==0)&(nz2==0) % -X side
ecm1(2)=1; % Activate!
end
case 3 % +Y case
if (nz2==0)&(nz4==0) % +Y side
ecm1(3)=1; % Activate!
end
case 4 % -Y case
if (nz1==0)&(nz3==0) % -Y side
ecm1(4)=1; % Activate!
end
case 5 % +Z case
if (length(find(NZ>0))>=2) % Two or more corners are above 0
ecm1(5)=1; % Activate!
end
case 6 % -Z case
if (length(find(NZ<0))>=2) % Two or more corners are below 0
ecm1(6)=1; % Activate!
end
```

```
end
ECM(ce,:) = ecm1;
end
% ECM should now be completely updated to reflect the structural
% membership
```

B.68 function vartoeud(variable, handle)

```
% function vartoeud(variable,handle)
% Purpose: VARIABLE goes to tag's USERDATA and STRING
set(handle,'UserData',variable);
if ischar(variable)
set(handle,'String',variable);
else
set(handle,'String',num2str(variable));
end
```

B.69 function ve(variable,handle);

```
% function ve(variable,handle);
% Purpose: Take variable and put it in the String of Handle
set(handle,'String',num2str(variable));
```

B.70 function viewbox(value)

```
global zoomout elslide azslide coord main_screen
% Front|Back|Top|Bottom|Left|Right|PRU|PRD|PLU|PLD|NRU|NRD|NLU|NLD
switch(value)
case 1
V = [90 0];
case 2
V = [270 0];
case 3
V = [90 90];
case 4
V = [90 -90];
case 5
V = [0 0];
case 6
V = [180 0];
case 7
V = [135 45];
case 8
V = [135 -45];
case 9
V = [45 45];
case 10
```

```
V = [45 -45];
case 11
V = [225 45];
case 12
V = [225 -45];
case 13
V = [315 45];
case 14
V = [315 -45];
otherwise
V = [135 30];
end
vaz = V(1); vel = V(2);
[dist] = udtovar(zoomout); % Default distance to Camera
set(elslide,'Value',vel); set(azslide,'Value',vaz);
moveabsolute(coord,18,vaz,vel);
moveabsolute(main_screen,dist,vaz,vel);
```

B.71 function viewer(action)

```
% function viewer(action)
global view_def elslide azslide refresh1 coord zoomout zoomin...
az el manual resolution zoom_inc main_screen
if isempty(main_screen),
['main_screen is empty in viewer']
end
switch(action)
case 'view_def'
value = get(view_def,'Value'); % Corresponds to a row in the listbox
viewbox(value); % 13 cases, so I put it in a function
case {'elslide','azslide'}
[dist] = udtovar(zoomout); % Default distance to Camera
[vaz,vel] = azel_slider(elslide,azslide); % Az and El from sliders
moveabsolute(coord,18,vaz,vel); % New look at 'coord'
moveabsolute(main_screen,dist,vaz,vel); % New look at 'main_screen'
case 'refresh1'
axes(main_screen); % set current axes to m_s
refresh % re-draw main_screen
case {'zoomout','zoomin'}
[dist_old] = udtovar(zoomout); % old default distance to camera
increment = ev(zoom_inc); % get value from editbox
switch(action)
case('zoomout') % move out of scene
dist_new = (dist_old - increment); % Zoom out
case('zoomin')
dist_new = (dist_old + increment); % Zoom in
otherwise
['Case zoomout/zoomin was not valid, so no action taken in viewer.m']
end
[vaz,vel] = azel_slider(elslide,azslide); % get az,el from sliders
vud(dist_new,zoomout); % Put dist in zoomout UserData
```

```
moveabsolute(main_screen,dist_new,vaz,vel); % New look at 'main_screen'
case 'az'
range_check(az,0,360,90);
etoud(az); % String to UserData of 'az'
case 'el'
range_check(el,-90,90,0);
etoud(el); % String to UserData of 'el'
case 'manual'
vaz = ev(az); vel = ev(el);
[dist] = udtovar(zoomout); % Default distance to Camera
set(elslide,'Value',vel);
set(azslide,'Value',vaz);
moveabsolute(coord,18,vaz,vel); % ""
moveabsolute(main_screen,dist,vaz,vel); % ""
case 'resolution'
range_check(resolution,1,17,10);
[res] = ev(resolution); % EditBox to Variable
set(elslide,'SliderStep',[res/180,0.1]);
set(azslide,'SliderStep',[res/180,0.1]);
case 'zoom_inc'
range_check(zoom_inc,1,100,10);
etoud(zoom_inc);
end
```

B.72 function vud(variable,handle)

```
% function vud(variable,handle)
% Send a Variable to UserData of Handle
set(handle,'UserData',variable);
```

BIBLIOGRAPHY

[1] Aceti, R. and G. Drolshagen. "Micrometeorites and Space Debris - The Eureca Post-Flight Analysis," *European Space Research and Technology Centre (ESTEC),* Noordwijk, Netherlands, (Nov 1994).

[2] Aero-Astro, *Bitsy Data Sheet*, World Wide Web, www.newspace.com/Industry/AeroAstro/.

[3] Air Force Research Laboratory TechSat 21 program, *Advanced Research and Technology Enabling Distributed Satellite Systems*, World Wide Web, www.vs.afrl.af.mil/VSD/TechSat21/.

[4] Allan, Bennet et al., *Crystals*, New York: Walker and Company, 1965.

[5] Beer, Ferdinand P. and E.R. Johnston, *Vector Mechanics for Engineers*, New York: McGraw Hill Publishing Company, 1988.

[6] Beni, G. and P. Liang. "Pattern Reconfiguration in Swarms–Convergence of a Distributed Asynchronous and Bounded Iterative Algorithm," *IEEE Transactions on Robotics and Automation*, v12, N3, June 1996.

[7] Boden, Margaret A., *The Philosophy of Artificial Life*, Oxford University Press Inc. New York, 1996.

[8] Bonabeau, E., G. Theraulaz, E. Arpin, and E. Sardet. "The Building Behavior of Lattice Swarms," In*: Proc. Artificial Life IV (*by Brooks, R. and Maes, P., eds.*)*, 307-312, MIT Press, 1994.

[9] Bonabeau, E. "The Design of Complex Architecture by Simple Agents," *Santa Fe Institute*, Santa Fe, NM., 1997.

[10] Bonabeau, E., et al. "Routing in Telecommunications Networks with 'Smart' Ant-like Agents," *Santa Fe Institute*, Santa Fe, NM., 1997.

[11] Bonabeau, E., et al. "The Emergence of Pillars, Walls, and Royal Chambers in Termite Nests," *Santa Fe Institute*, Santa Fe, NM., 1997.

[12] Bonabeau, E. "Marginally Stable Swarms Are Flexible and Efficient," *J. Phys*, v6 (1996) pp. 309-324.

[13] Bonabeau, E. "Self-Organization in Social Insects," *Trends in ecology and evolution,* v12, N5 (May 1997).

[14] Bonabeau, E., A. Sobkowski, G. Theraulaz, and J.-L. Deneubourg. "Adaptive Task Allocation Inspired by a Model of Division of Labor in Social Insects." Santa Fe Institute, Santa Fe, NM., 1997.

[15] Burks, A.W., ed., *Essays on Cellular Automata,* Illinois: University of Illinois Press (1968) p. xv.

[16] Codd, E.F. *Cellular Automata,* New York: Academic Press, 1968.

[17] Cobb, Richard G. *Structural Damage Identification from Limited Measurement Data,* PhD Dissertation, AFIT/DS/ENY/96-3, School of Engineering, Air Force Institute of Technology,

Wright-Patterson AFB, OH, (Mar 1996).

[18] Committee on Advanced Robotics for Air Force Operations, AFSB, and NRC., *Advanced Robotics for Air Force Operations*, National Academy Press, Washington D.C., 1989.

[19] Culik II, K. and L.P. Hurd. "Computation Theoretic Aspects of Cellular Automata," *Physica D*, v45 (1990) pp. 357-378.

[20] Delgado, J. and R.V. Sole'. "Self-Synchronization and Task Fulfilment in Social Insects," *Department de Llenguatges i Sistemes Informatics, Universitat Politecnica de Catulunya,* Barcelona Spain (submitted to *Proc. Roy. Soc. B*), 1998.

[21] Dellaert, F. and R.D. Beer. "A developmental model for the evolution of complete autonomous agents," In P. Maes, M. Mataric, J. Meyer, J. Pollack and S. Wilson (eds.), *From Animals to Animats IV: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, 393-401, MIT Press, 1996.

[22] Deneubourg, J.-L., G. Theraulaz, and R. Beckers. "Swarm-Made Architecture," *In: Toward a Practice of Autonomous Systems, Proceedings of The First European Conference of Artificial Life (*Varela, F.J. and Bourgine, P., eds*),* 123-133, Cambridge, MA: The MIT Press/Bradford Books.

[23] Grey, J. and L.A. Hamdan. "Space Manufacturing IV," *Proceedings of the Fifth Princeton/AIAA Conference* May 18-21, 1981.

[24] Hölldobler, B., and E. O. Wilson, *The Ants*, Cambridge: Belknap Press, 1990.

[25] Jilla, C.D. and D.W. Miller. "Satellite Design: Past, Present and Future," *International Journal of Small Satellite Engineering*, (12 Feb 1997).

[26] Jones, L. Joseph and Anita M.Flynn, *Mobile Robots: Inspiration to Implementation.* Massachusetts: A K Peters Wellesley 1993.

[27] Karsai, I., Z. Penzes, and J.W. Wenzel. "Dynamics of Colony Development in Polistes-Dominulus - a Modeling Approach," *Behavioral ecology and sociobiology,* v39, N2 (Aug 1996).

[28] Karsai, I. and G. Theraulaz. "Nest-Building in a Social Wasp--Postures and Constraints (Hymenoptera, Vespidae)," *Sociobiology,* v26, N1, 1995.

[29] Langton, C.G. "Self-Reproduction in Cellular Automata," *Physica 10D* (1984) pp..135-144.

[30] Langton, C.G., N.H. Packard, and Wentian Li. "Transition Phenomena in Cellular Automata Rule Space," *Complex Systems Group, Theoretical Division, Los Alamos National Laboratory, Physica D* v45 (1990) pp. 77-94.

[31] McIntosh, Harold V. "Wolfram's Class IV Automata and a Good Life," *Physica D* 45 (1990) pp. 105-121.

[32] Miller, S.L. and C.U. Harold. "Organic Compound Synthesis on the Primitive Earth," *Science*, v130, N3370 (31 July 1959).

[33] Minagawa, M. "Solving Block Stacking Problems in Cellular Space," *Sapporo Gakuin University, Japan,* 1993.

[34] Miramontes, O., R.V. Sole', and B.C. Goodwin. "Collective Behavior of Random-Activated Mobile Cellular Automata," *Physica D*, v63 (1993) pp. 145-160.

[35] Rauch, E.M., M.M. Millonas, and D.R. Chialvo. "Pattern Formation and Functionality in Swarm Models," *Physica Letters A*, v207 (1995) pp. 185-193.

[36] Reynolds, Craig W. "Flocks, Herds, and Schools: A Distributed Behavioral Model." *In Proceedings of* SIGGRAPH '87 (*Computer Graphics, Annual Conference Series*), 25-34, 1987.

[37] Rosheim, Mark E., *Robot Evolution*, New York: John Wiley and Sons, Inc. (1994) pp. 275-280.

[38] Rus, Daniela. "Self-Reconfiguring Robots," *IEEE Intelligent Systems*, 1998.

[39] Rylaarsdam, Jillene B. *International Space Station traffic modeling and simulation.* MS thesis, AFIT/GOA/ENS/96M-08, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH, Dec 1996.

[40] Silva Curiel, R.A., *Small Satellite Home Page (SSHP)*, World Wide Web, www.ee.surrey.ac.uk/EE/CSER/UOSAT/SSHP.

[41] Sims, Karl. "Evolving Virtual Creatures," *In Proceedings of* SIGGRAPH '94 (*Computer Graphics, Annual Conference Series*) (1994) pp. 15-22.

[42] Sims, Karl. "Evolving 3D Morphology and Behavior by Competition," *Artificial Live IV Proceedings,* (ed. by R. Brooks and P Maes, MIT Press), 28-39, 1994.

[43] Sims, Karl. "Karl's computer circus," *Science*, (Aug 17 1994).

[44] Technion - Israel Institute of Technology, Asher Space Research Institute, *A High Resolution Multispectral Remote-Sensing Microsatellite (TechSAT IIa)*, World Wide Web, www.techion.ac.il/pub/projects/techsat/tech2.html.

[45] Theraulaz, G. and Eric Bonabeau. "Coordination in Distributed Building," *Science,* v269 (1995) pp. 686-699.

[46] Theraulaz, G. and Eric Bonabeau. "Modeling the Collective Building of Complex Architectures in Social Insects with Lattice Swarms," *J. theor Biol.* (in press), 1995

[47] Theraulaz, G. Gervet, and J. Tianchanski, SS. "Social Regulation of Foraging Activities in Polistes Dominulus Christ - a Systematic Approach to Behavioral Organization," *Behavior,* v116, (Mar. 1991).

[48] Theraulaz, G., S. Goss, J. Gervet, and J.-L. Deneubourg. "Task differentiation in *Polistes* wasp Colonies: a Model for Self-Organizing Groups of Robots," In: *From Animals to Animats*, Proc. of the 1st International Conf. on Simulation of Adaptive Behavior (Meyer, J.A. and Wilson, S.W., eds), 34-355, MIT Press, 1991.

[49] Thompson, A. and P. Layzell. "Analysis of Unconventional Evolved Electronics," *Centre for Computational Neuroscience and Robotics*, University of Sussex, Brighton, UK., 1998.

[50] Neumann, John von, *The Theory of Self-Reproducing Automata*, Illinois: University of Illinois Press, 1966.

[51] Neumann, John von, *John von Neumann collected works*, Pergamon Press, v5 (1963) pp. 288.

[52] Wagner, Israel A. "Cooperative Covering by Ant-Robots using Evaporating Traces," *Department of Computer Science* Technion City, Haifa 32000, Israel, July 26, 1996.

[53] Webb, Barbara. "A Cricket Robot," *Sci. Am.*, Dec 1996.

[54] Wilson, E.O., *Animal Behavior*, v10 (1962) pp. 134-164.

[55] Wilson, E.O., *The Insect Societies*, Cambridge: Belknap Press, 1971.

[56] Wolfram, S. "Statistical Mechanics of Cellular Automata." Rev. Mod. Phys. v55 (1983) pp. 601.

[57] Wolfram, S. "Universality and Complexity in Cellular Automata." Physica D, v10 (1984) pp. 1.

[58] Wuensche, A. "Classifying Cellular Automata Automatically," *Santa Fe Institute*, Santa Fe, NM, 1997.

Vita

Lieutenant Daniel J. Petrovich was born on 17 February 1975 in Kalispell, Montana. He graduated from Hellgate High School in Missoula, Montana in 1993. He then attended the University of Rochester in Rochester, New York where he received a B.S. in Electrical Engineering/Computer Architecture with High Distinction. Upon graduation, he accepted a commission as a Second Lieutenant in the United States Air Force. In June, 1997, Petrovich was accepted for Direct Accession to the Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio to pursue a Master of Science degree in Electrical Engineering with a primary specialty in Communications and a secondary specialty in Automatic Target Recognition. Upon completion of that assignment in March, 1999, Lieutenant Petrovich will be assigned to the Air Force Research Laboratory at Wright-Patterson Air Force Base, Ohio.